PGRemote
User's Manual

Table of Contents

Contacting **The *Moving Pixel* Company**

Phone	+1.503.626.9663	US Pacific Time Zone
Fax	+1.503.626.9653	US Pacific Time Zone
Address	**The *Moving Pixel* Company** 4905 SW Griffith Drive, Suite 106 Beaverton, Oregon 97005 USA	
Email	information@movingpixel.com	
Web site	http://www.movingpixel.com	

Documentation

1 Overview

PGRemote application software is one component of a joint MIPI test solution offered by **The *Moving Pixel* Company** (TMPC) and Tektronix. In this solution, TMPC software and hardware are used to provide stimulus data traffic to a target MIPI device while a Tektronix logic analyzer is used to capture, decode, and display bus activity using a custom bus support.

This document describes the PGRemote software and operation of the TMPC MIPI solution, which makes use of the following components:

1) PGRemote application: Windows application that embodies knowledge of the MIPI protocol suite including DCI, CSI, DSC, DCS and D-PHY to build signaling and command data for a general-purpose pattern generator (PG) to play out on the DPhy bus.[1]

2) PGAppDotNet software: Windows control application for the PG3A pattern-generator module with the capability to receive commands remotely via .NET. This software defines data blocks and playback sequences for general-purpose data output and, once defined, downloads this information to the PG3A hardware over a USB link, and manages real-time playback.

3) PG3A pattern-generator module: stand-alone or TLA plug-in hardware for data output. The PG3A module is a broadly generic and flexible instrument for generating arbitrary data patterns, using separate probe accessories to support a variety of digital standards and protocols.

4) P331 probe: this is TMPC's basic probe designed expressly for MIPI/DPhy signaling.[2] The P331 probe has the following features:

    - Supports up to 4 data lanes plus a clock lane, each up to 1Gbps.
    - Uses two PG3A probe connections (port A and B)
    - SMA outputs for each lane
    - Proper LP-to-HS protocol on clock lane
    - Clock lane can run continuously, even when PG3A is stopped
    - Provides an optional external clock input (Opt. Clock In) port suitable for impaired clock testing (e.g. jitter, noise, etc), where output data is clocked using a filtered version of the impaired clock.
    - Proper BTA response handshaking when DUT returns control back to PG
    - LP contention detection

[1] A newer variation of PGRemote called PGRemoteForP338 is now available that supports the P338 probe. Except where explicitly noted, references to PGRemote include PGRemoteP338 as well.

[2] The next-generation probe to the P331 is the P332, which supports up to 1.5 Gbps per lane. Except where explicitly noted, references to the P331 include the P332 and P338 probe types as well.

- Ability to insert simple commands into vertical blanking while in continuous video mode
- Adjustable HS voltages and delays for each lane (which have a common LP voltage) with support for real-time adjustment during setup

5) Inputs probe: a probe used to provide an external clock input (and events) to the PG3A module. When a P331 is used, the Inputs probe is necessary to connect a master clock from the P331 to the PG3A and the PG3A always operates in "External" clocking mode.[3]

Figures 1 shows the overall test architecture when using the P331 probe.

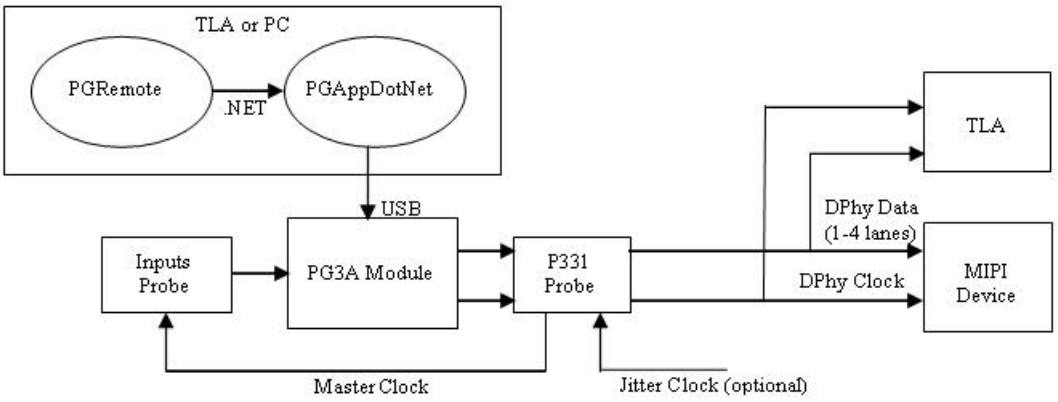

Figure 1 – MIPI Test Architecture (P331)

[3] While the PG3A always operates in External clocking mode when using the P331 probe, the probe itself can operate in Internal or External clocking mode. So no functionality is lost.

2 Setup & Installation

This document assumes general familiarity with the PG and PGApp. If necessary, please read the *PGUsersManual* for more information before continuing.

2.1 Installing PGAppDotNet

A special version of PGApp software is needed for MIPI testing, named PGAppDotNet. It has been modified to support a more easily used (by TMPC) remote communication protocol via .NET rather than the PPI interface provided via COM/DCOM. Otherwise, PGAppDotNet behaves similarly to PGApp for general use (though system files between the two are *not* compatible). It is no problem to have both PGApp and PGAppDotNet installed on the same machine at the same time.

Installation of PGAppDotNet is straightforward. Follow the instructions below:

1) If you already have a previous version of PGAppDotNet installed, it must first be uninstalled. You may do this via Windows: Start->Settings->Control Panel->Add or Remove Programs, scroll to PGAppDotNet and click, then click the Change/Remove button and follow the instructions.
2) Run the new setup file, SetupDNXXX.exe where XXX is the application build number (currently 019). If you do not have an installation disk or a setup file for PGApp, you may download the most recent version from www.movingpixel.com. Follow the installation instructions which by default will install the application in the folder: **c:\Program Files\TMPC\PGAppDotNet**.
3) If you are planning to run on a real PG module (as opposed to just demonstrating in offline mode), you must obtain a TMPCLicense.txt file from the Moving Pixel Company and copy this file into the PGAppDotNet application directory (i.e. **c:\Program Files\TMPC\PGAppDotNet\PGAppDotNet**). See additional comments at the end of this section.
4) Run PGAppDotNet and check that you have the correct executable: use the menu option "Help->About PGApp" to show the version and additionally display "(Dot Net Server Version)".

IMPORTANT: Depending on your hardware configuration and the DPhy protocols you wish to support, one or more license options must be enabled for your PG3A module for proper operation. Options are enabled with license strings provided by TMPC and are derived from the module serial number and located in the file TMPCLicense.txt. You will need to obtain this license file from TMPC before using the system with real hardware.

If you are using the P331 probe, you only need the protocol option(s) enabled (i.e. DSI or CSI); no DDROutput option is required.

To verify that licensing options are properly enabled on your module once you've installed the PGAppDotNet application, power on the PG3A, connect to PG module, and

use the menu "System->System Properties" to bring up the properties window and see the options listed.

Note that if PGAppDotNet is running in Offline mode (i.e. is not connected to a physical PG module), all MIPI protocol options are automatically enabled for PGRemote demonstration.

2.2 Installing PGRemote

The PGRemote software is generally installed on the same computer as PGAppDotNet (but it is not required as long as both computers are connected via a LAN). To install PGRemote, simply execute the PGRemote setup file and step through the setup windows. The PGRemote application is installed by default in the **c:\Program Files\TMPC\PGRemote** directory.

One requirement for running the PGRemote application is the Microsoft .NET Framework 3.5 platform. You can check if .NET 3.5 is installed through **Start->Settings->Control Panel** and double-clicking "Add or Remove Programs". Scroll through the list of applications for "Microsoft .NET Framework 3.5". If version 3.5 is present, so will versions 3.0, 2.0 and 1.1.

If you are installing from an installation CD provided by The Moving Pixel Company, Microsoft .NET 3.5 will be installed automatically (if necessary). On the other hand, if you are running the PGRemote installation from a setup file downloaded from our website or received via email, the setup files for installing .NET 3.5 will not be present.

In this case, running the PGRemote setup on a machine without .NET 3.5 installed will appear to begin .NET installation (asking the user to accept supplemental license terms), but then will subsequently report an error installing system components. If this occurs, the user should close the error report and manually install .NET 3.5.

To install Microsoft .NET 3.5 (without an installation CD from The Moving Pixel Company) your computer must be connected to the internet. The following is a link to the installation executable (note: this file is over 200 MB):

http://download.microsoft.com/download/6/0/f/60fc5854-3cb8-4892-b6db-bd4f42510f28/dotnetfx35.exe

Download the executable and run it and follow the setup instructions.

Finally, it has been discovered that .NET 3.5 installation may fail on some machines when the Windows IIS service is installed. It appears that some TLAs have this service installed by default even though it is not necessary for normal operation (the Windows IIS service is in support of making your computer a web server). Thus, if you cannot successfully install .NET 3.5, it may be necessary to uninstall the Windows IIS service before trying again. Please note that if re-installation of the Windows IIS service after .NET 3.5 installation is desired, this will require a Windows XP System CD.

To uninstall the Windows IIS service, do the following:

1) Select Start->Settings->Control Panel
2) Double-click "Add or Remove Programs"
3) Click on "Add/Remove Windows Components" in left pane
4) Uncheck the "Internet Information Service (IIS)" box
5) Click Next button
6) Click Finish button

Please be aware that if DotNet 3.5 is not installed on your machine, PGRemote will NOT run at all (nothing will happen at launch -- no windows, warnings, or errors will appear).

2.3 Setting up the PG3A Module and Probes

2.3.1 PG3A Installation

Install the PG3A module following instructions in the *PG3AUsersManual*, connecting the host computer to the module via a USB cable. For MIPI testing, a single P331 MIPI probe needs to be connected to the PG3A. Note, for the purposes of this document, references to the P331 probe also applies to the P332 probe as well (as the only difference beween the two probes is the maximum supported HS).

2.3.2 P331 Probe Connection

For a P331 configuration (see **Figure 1**), a single P331 probe needs to be connected to the A and B connectors of the PG3A. Connect the left connector of the P331 to connector A on the PG3A and the right connector of the P331 to connector B on the PG3A.

In this configuration, the P331 always sources the master clock to the PG3A and thus, an Inputs probe is always required. Use the CkP/CkN signals from the P331 as the input source to the Inputs probe and connect the probe to the PG3A. **Figure 2** shows the P331 back panel connectors for reference.

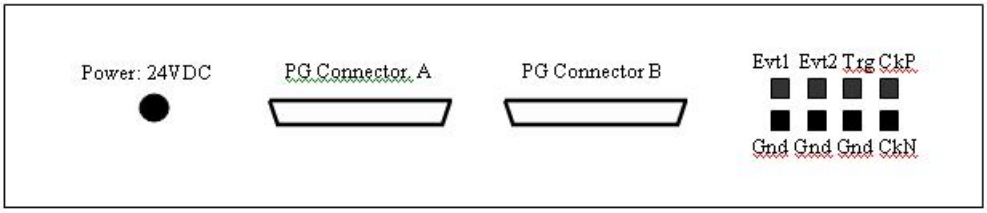

Figure 2 – P331 Back Panel

The event signals are available for user diagnostics and monitoring, and are currently assigned as follows:

- Evt1 – pulses high when a command is inserted into vertical blanking during video mode (see section 3.6.2)

- Evt2 – asserts high when any of the four LP contention states are detected (see section 0)
- Trig – reserved

On the front-panel of the P331 are eleven SMA connectors: five differential outputs for four data lanes and a clock, as well as the Opt. Clock input. The Opt. Clock input is connected when external clocking is desired, whether for impaired clock testing or to make DPhy lane timing synchronous with other devices. When "External" clocking is selected in the PG configuration dialog of PGRemote, the Opt. Clock is output directly as the DPhy lane clock, and, in addition, a filtered version of the Opt. Clock input is used for data clocking.

2.3.3 Clock/Data Timing Adjustment

Regardless which clocking scheme is used, the data and clock must be adjusted to have a specific alignment relative to each other. Generally, users want an ideal quadrature sampling relationship between the clock and data, where the lane clock is delayed by 90 degrees relative to the data. For example, a lane clock of 300 MHz (600 MBit/s) will likely want the clock delay to be set to around 830 ps to center the clock edge properly with respect to the HS data. Note that the maximum lane delay/skew adjustment is 2.4 ns for the P331. Thus, only at lane clock frequencies above 104 MHz can the ideal relationship be achieved, and below 104 MHz the timing range cannot cover all possible timing relationships.[4]

Adjusting clock lane timing relative to the data lanes is achieved using the delay adjustment controls in the PG Configuration window of the PGRemote application (section 3.4.1). The P331 probe supports a "Real-time Adjust Mode" activated from the PG Configuration dialog (see section 3.4.1**)** that causes all lanes (including clock) to output a high-speed 101010… clock pattern. When this mode is active, delay adjustments take effect immediately rather than only when the dialog is closed.

Thus, the procedure for skew adjustment is simply to connect the clock and one data lane (all data lanes should be aligned) to an oscilloscope (or using MagniVu capture on the TLA) and adjust the HS clock timing as desired. Generally, HS data lane timings do not need adjustment except for advanced testing. Note also that, for the P331, LP lane delays are the same as HS lane delays.

[4] When using external clocking with the P375 probe, end-to-end clock delays through the PG will position output data at an arbitrary phase relative to the external lane clock. Thus, delay/skew adjustment requirements to achieve ideal timing may be different than for internal clocking.

3 Using PGRemote

The PGRemote software is the principal application that users interact with to configure and send DSI/CSI commands to their DUT.

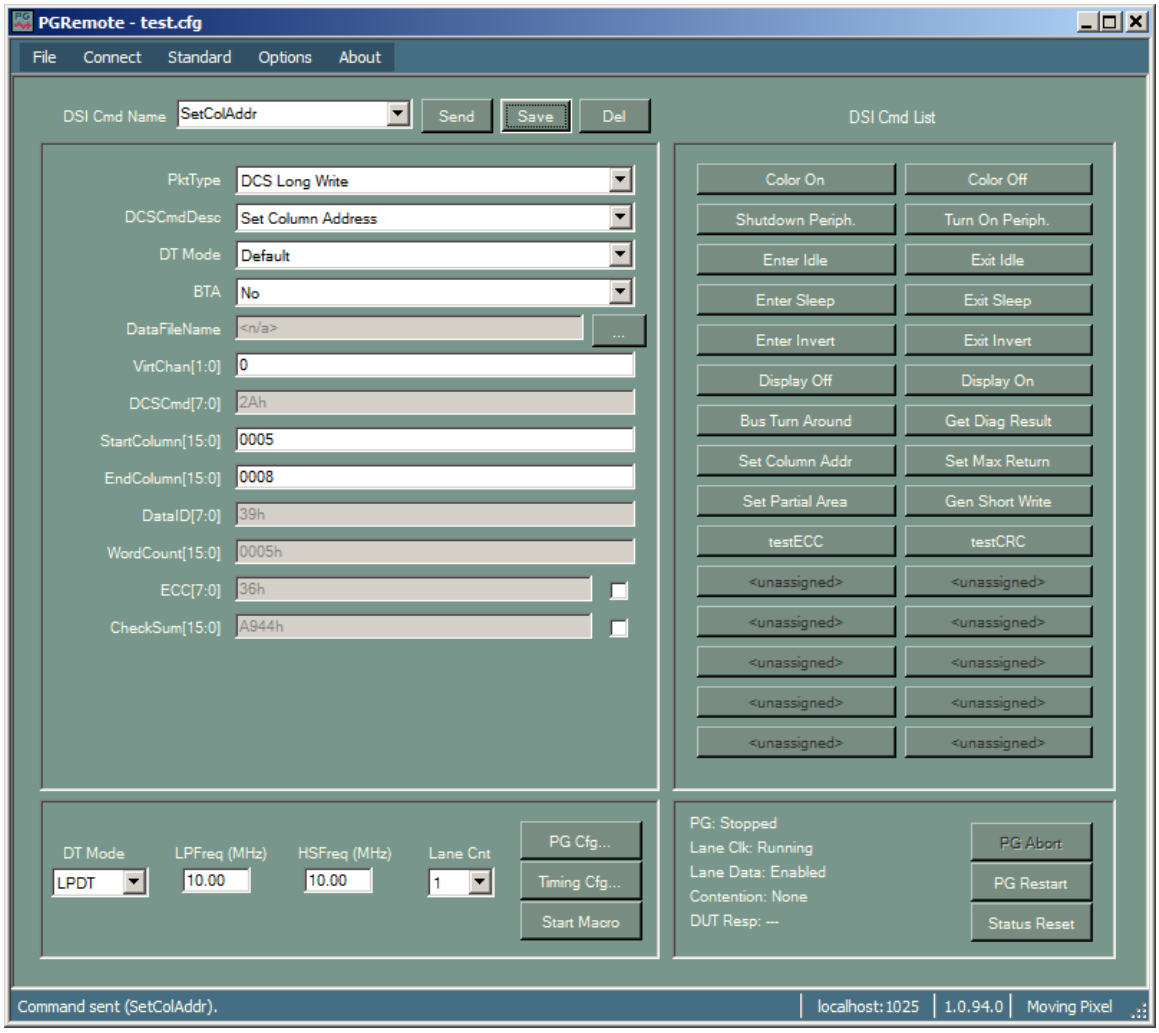

Figure 3 – PGRemote Main Window

3.1 Software Architecture Overview

Most of the interaction between the user and the MIPI test system occurs in the PGRemote main window (see **Figure 3**). It has several distinct sections.

- The left pane of the main window allows the user to define MIPI commands and their arguments, naming them to allow for later recall and sending to the PG, as well as assigning them to command buttons.

- The right pane of the main window consists of 30 command buttons that can each be individually associated with a defined command, allowing single-click sending of command commands to the PG.
- The bottom left pane of the main window provides controls for configuring parameters of PG playback, MIPI protocol definitions, and physical characteristics of the DPhy bus.
- The bottom right pane of the main window comprises PG and probe status and operational controls.
- The status bar at the bottom of the main window displays informational and error messages and indicates the current PG host connection.

The following sections describe more details of PGRemote operation.

3.2 Quick-Start Summary

The following summarizes the overall procedure for installing and configuring the TMPC software and hardware for MIPI testing:

1) Install and connect a PG3A module to a PC (via USB).

2) Attach a P331 probe to the A/B connectors of the PG3A.
3) Connect an Inputs probe to the PG3A and connect the CkP/CkN square pin outputs from the back panel of the P331 (see **Figure 2**) to the clock input of the Inputs probe.
4) If external clocking is to be used, connect an external clock to the Opt. Clock input on the front panel of the P331.
5) Connect the SMA data and clock lane outputs from the P331 to the DPhy lane inputs of the DUT (or scope if performing timing skew adjustments).
6) Apply power to the P331 probe.
7) Install the PGAppDotNet server application (section 2.1).
8) Obtain the appropriate TMPCLicense.txt file and copy it to the PGAppDotNet application directory (c:\Program Files\TMPC\PGAppDotNet\PGAppDotNet).
9) Install the PGRemote control application (section 2.2).
10) Start the PGAppDotNet server and connect to the PG3A module as the Master Module (File->Choose PG…).
11) Start the PGRemote application and connect to the PGAppDotNet server (section 3.3)
12) Configure PG operational parameters (section 3.4.2).
13) Configure options for the PG, e.g. voltage levels (section 3.4.1)
14) Set and/or adjust the clock/data timing skew relationship (section 2.3.3).
15) Configure video timing parameters for your device (section3.6.3)
16) Define commands and assign them to buttons, if desired (section 3.7).

Once these steps are completed, the system is ready to start testing.

For demonstration without a real PG3A module, a simplified procedure can be used starting at step 7 above, with the following modifications:

1) Step 8 may be skipped. When running in offline mode, all MIPI protocol options are automatically enabled.
2) In step 10, when connecting to a PG module, select "Offline" as the Master Module.
3) In step 11, if you are running recent versions of PGAppDotNet and PGRemote, PGRemote will automatically configure offline modules to have a P331. Otherwise, you need to manually configure your offline module to have a P331 probe as follows. Go to the Probe Setup Window and set the probe type of the A probe to P331. To do this, in the System Window, click on the Setup button, then the Probe Setup tab, and then click in the A0 Type box in the grid window to bring up a drop-down selection box. Choose P331.

3.3 Connecting to the PGAppDotNet Server

Before any MIPI commands can be sent, the PGRemote application needs to connect to the PGAppDotNet server, which must be launched before initiating a connection within PGRemote. The appropriate license option (DSI or CSI) must be enabled for your PG3A module for the PGAppDotNet server to respond to incoming requests from PGRemote (see section 2.1 for details).

Selecting the "Connect" menu option brings up a connection dialog with a textbox to enter in the name of the machine where the PGAppDotNet server is running (if both PGRemote and PGAppDotNet are running on the same machine the user can leave the textbox blank). After typing the host name, click on the Connect button to connect to the server.

An alternate format for the host name can include an IP port number. The syntax in this case is "<hostName>:<portNum>" (note the colon character). Each PGAppDotNet server opens a unique IP port number to use for incoming requests. The default behavior of PGAppDotNet is to first try port number 1025, and, if this fails, try port number 2798. PGRemote PGRemote mirrors this behavior when the user asks to connect to the server: first try 1025, then 2798.

Note, in the more unusual case where multiple servers are brought up on the same host machine, to bring up a the second corresponding copy of PGRemote, the user must specify a proper secondary port number (otherwise, the second copy will connect to port 1025 like the first copy).

If the connection is successful, the dialog will close and the host name (including port number) will be displayed in the center panel on the status bar indicating the current connection. In addition, the PGAppDotNet server will show "Client Connected" in the center panel of its status bar.

If the connection is unsuccessful, the problem may be among the following:

1) The host name and/or port number is not correct

2) The host is not accessible via the network from the current computer
3) The PGAppDotNet server is not running
4) The non-server version, PGApp, is running instead of the PGAppDotNet server. To verify, go to the "Help->About PGApp" menu and see that the third line in the dialog says "(Dot Net Server Version)".

The "Disconnect" menu is used to close the server connection when testing is complete. The application will do this automatically on closing so it is not required. Once connected, the center panel of the status bar will read "Offline", indicating no connection to a PGApp server.

Note that on subsequent restarts of PGRemote, the application automatically attempts to reconnect to the last PG host used previously. You can tell if reconnection was successful by the second pane of the status window. If "Offline" is displayed, connection was unsuccessful.

3.4 PG Configuration

3.4.1 PG Configuration Dialog

Once connected, the PGAppDotNet server and P331 probe need to be configured with parameters specific to your MIPI testing needs. This is done via the PG Configuration dialog brought up from the main window via the "PG Cfg..." button in the lower left pane of the main window.

After entering desired settings on the dialog, click the OK button to accept them or Cancel if you want to discard them. Clicking on the "Defaults" button sets all controls to their default values for the current probe type selected.

Unless PGRemote is in "real-time adjust mode", configuration values are only sent to the PG after the dialog is closed with the OK button.[5] They are also automatically saved and restored as application settings when PGRemote is closed and restarted. Values are subsequently sent to the PG on reconnection to the PGAppDotNet server. Note that PGRemote overrides any manual adjustments the user has made to these settings in the variable probe window of PGAppDotNet.

Figure 4 shows the PG Configuration dialog. Below is a description of its controls:

[5] Clicking the Cancel button in the PG Configuration Dialog will, in general, discard any changes and not configure the PG and probe. However, if changes were made in real-time adjust mode, clicking the Cancel button will configure the PG and probe, restoring settings back to their original state.

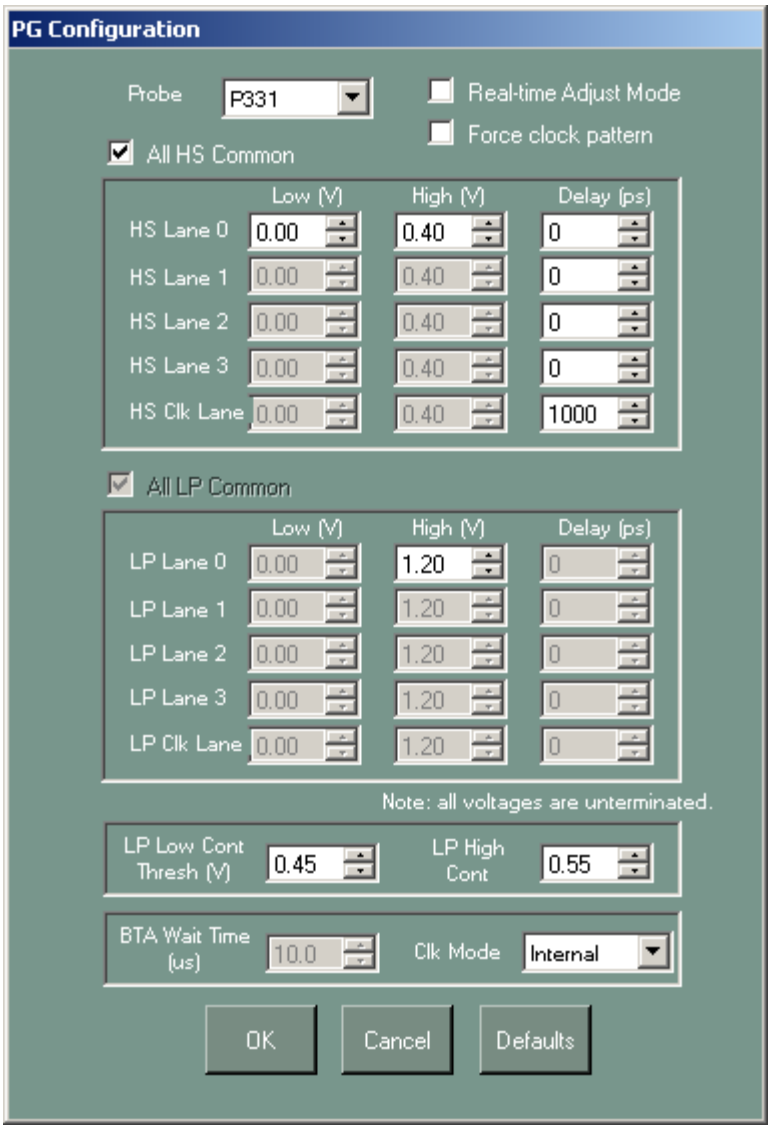

Figure 4 – PG Configuration Dialog

Probe Type: At the top is a drop-list control for the user to indicate which probe type is to be used for this session. As the probe type is central to entire system operation, this selection affects controls and functions throughout the PGRemote application (e.g. voltage/delay limits, frequency range, etc.). Thus, the user should always select the probe type first.[6] Since the use of the P375 probe is now unsupported, only the P331 probe is currently available in this control.

Real-time Adjust Mode: this check box, when checked, cause voltage and delay adjustments to take effect immediately rather than wait until the dialog is closed. This

[6] Because PGRemote can operate without a server connection, it must be told what the probe type is rather than query the PGAppDotNet server. Clearly, the probe type setting must match the actual probe type connected to the PG3A and PGRemote will notify the user when the PG is configured that there is a probe type mismatch.

mode is often used with a looping command or video (or the Force Clock Pattern control) while monitoring the MIPI signals with a scope. The procedure to do this is as follows:

1) Send a video command or loop a non-video command.
2) While the PG is running, click on the PG Cfg... button to bring up the PG Configuration dialog.
3) Click on the "Real-time Adjust Mode" check box.
4) When asked whether to stop the PG, click No.
5) Now any adjustments you make will take effect immediately.
6) When you are done with all your testing, click Cancel on the dialog (otherwise, clicking OK will stop the PG).

Force Clock Pattern: checking this box causes a test clock pattern lane (at the current HSFreq setting in the main window) to be output on all P331 probe lanes (plus the clock lane). This pattern is generally used in conjunction with real-time adjustment and an oscilloscope to visually adjust the setup and hold time of the clock lane with respect to the data lanes at the DUT.

All HS Common: this check box, when checked, disables the HS voltage controls for all lanes except for data lane 0. Adjustments to data lane 0 HS voltages are automatically applied to all lanes. Otherwise, HS voltage settings can be independently adjusted.

HS Low/High Voltage Controls: these controls set the unterminated HS voltage levels for data and clock lanes. When connecting to real hardware, HS voltages generally should be set to 0.0V (low) and 0.4V (high). Slave termination resistance will reduce HS levels to the expected swing of 0.1 - 0.3V. For the P331 probe, voltage limits are -0.6V to 1.2V.

HS Delay Controls: these controls set the relative delays of each lane. Delay adjustments are in ps and have a range of 0 to 2400 ps. For the P331 probe, HS and LP delays are common, so adjusting the HS lane delay sets the LP lane delay and vice-versa. Note that the HS (and LP) delay control for the P331 clock is disabled when in external clocking mode. In this mode, clock delay adjustments relative to the data can only be made by adjusting the all the data lane delays by the same amount.

All LP Common: this check box, when checked, disables the LP voltage controls for all lanes except for data lane 0. Adjustments to data lane 0 LP voltages are automatically applied to all lanes. Otherwise, LP voltage settings can be independently adjusted. For the P331 probe, this box is always checked.

LP Low/High Voltage Controls: these controls set the LP voltage levels for data and clock lanes. When connecting to real hardware, LP voltages generally should be set to 0.0V (low) and between 1.1V and 1.3V (high). For the P331 probe, the low voltage is always set to 0.0V and the high voltage limits are 0.45V to 3.6V.

LP Delay Controls: these controls set the relative delays of each lane. Delay adjustments are in ps and have a range of 0 to 2400. In the P331 probe, HS and LP delays are common, so adjusting the LP lane delay sets the HS lane delay and vice-versa. Note that the LP (and HS) delay control for the P331 clock is disabled when in external clocking mode. In this mode, clock delay adjustments relative to the data can only be made by adjusting the all the data lane delays by the same amount.

LP Contention Thresholds: these two controls are enabled only for the P331 probe and determine the threshold voltage used for contention detection.

The LP Low contention threshold determines the voltage at which a LP Low fault is flagged (i.e. when the transmitter attempts to drive an LP0 and measures a voltage on the line greater than the threshold). This value is nominally 0.45V according to the DPhy specification.

The LP High contention threshold determines the voltage at which a LP High fault is flagged (i.e. when the transmitter attempts to drive an LP1 and measures a voltage on the line less than the threshold). This value is nominally 0.55V according to the DPhy specification.

BTA Wait Time: this legacy control is always disabled since the P375 probe is no longer supported for MIPI. Note that the P331 probe responds properly to BTA return sequences so this field is unused.

Clk Mode: selects the clocking mode of the PG or probe. For the P331 probe, this setting determines whether the lane clock is internally generated (Internal mode) or sourced by the Opt. Clock input of the probe (External mode). In external mode, the Opt. Clock input is passed unaltered to the lane clock output and is filtered to for use in clocking data output, allowing for the use of an impaired clock to be used for testing. Note that regardless of clock mode setting, the PG is configured to run in external clocking mode, as it receives a gated 300 MHz clock from the P331 probe via the Inputs probe.

3.4.2 PG Configuration Controls in Main Window

In addition to the settings from the PG Configuration dialog, four controls at the bottom of the main window are used to describe DPhy configuration parameters. These controls are described below:

DT Mode: this combo box sets the default data transfer mode for commands whose DT Mode field is set to "Default". Packets may be sent as either HS packets (HSDT mode) or LP packets (LPDT mode). Individual commands can override the default DTMode setting by setting the DTMode field in their command arguments. Note: this mode only applies to non-video commands and does not affect video mode operation or low-level DPhy commands such as BTA, Trigger, etc.

HSFreq: this control sets the lane clock frequency of the DPhy bus. If clocking is external, this value is still necessary to inform the software what frequency the external clock will be running at (additionally, the P331 probe must have this information to properly condition its Opt. Clock input).

Note that since HS data transitions on both clock edges, the bit rate of each lane will be 2 * HSFreq and the total bit-rate of the link will be (LaneCnt * 2 * HSFreq). For the P331, the maximum frequency is 580 MHz (1.16 Gbps)[7] and the minimum frequency is 1 MHz (internal) or 25 MHz (external). The maximum P332 frequency is 750 MHz (1.5 Gbps).

An exception to the normal behavior of this control is when the user is either using the CSI command set or is using the DSI command set and has set the Burst Mode to "Non-burst" in the current timing configuration (see section 3.6.1). In these cases, whenever the current displayed command is a video command, the HS Freq control becomes read-only and reflects the frequency that will be used when the command is sent to the PG.[8] The computed frequency takes into account all of the current timing settings as well as the lane count.

Finally, because LP and HS data are stored together as PG vectors and, in the case of the P331, LP and HS bits are multiply packed into vectors, the frequency of LP bits cannot be set independent of the frequency of HS bits. Thus, whenever the HSFreq setting is changed, the LPFreq setting is newly quantized to the nearest legal value and a message is displayed in the status window.

LPFreq: this control sets the frequency that LP bits are output onto the DPhy bus (i.e. determines T(LPX) in the DPhy standard). As the PG vector rate is based on HSFreq, setting LPFreq effectively determines the number of times each LP data bit is replicated. Because of this, as discussed in the description for HSFreq, whenever the LPFreq setting is set, the value is quantized to the nearest legal value based on HSFreq.[9]

Lane Cnt: this control selects the number of DPhy lanes to use (1-4). HS data is demultiplexed onto DPhy lanes according to the protocol specification. Unused lanes are held in the LP11 output state.

3.5 DPhy Timing Configuration

The DPhy Timing Configuration dialog is brought up via the "Config DPhy Timing" menu item in the Options menu. This dialog allows the user to set DPhy timing parameters, most pertaining to the timing of clock and data transitions from LP to HS and from HS to LP mode. In general, the default DPhy parameter values computed by the

---

[7] Requires P331 probe firmware version 1.4 or greater. Otherwise, the maximum bit rate for the P331 probe is 1 Gbps.

[8] Note this behavior also occurs when the mouse hovers over a command button that is assigned to a video command and "Display Command on Button Hover" is checked in the Options menu.

[9] The formula for LPFreq quantization is dependent on probe type and HSFreq. It can take one of the forms: HSFreq/1 (at low frequencies), HSFreq/2 (P375 use or moderate frequencies), or HSFreq/4 (at high frequencies)

system are sufficient and customizing these values is unnecessary. However, some users would like to have the ability to set these parameters more specifically.

DPhy parameter values and the settings from this dialog are saved as application settings when PGRemote is exited. As with other application settings, they are restored when the application is restarted.

Figure 5 shows the dialog's simple layout, with a Configuration drop-down box for selecting among timing parameter sets, a left column of text boxes for viewing/setting the parameters, and a right column of read-only text boxes for viewing the quantized parameters actually used by the PG. Values are quantized for use by the system based on the current HS clock frequency and is required due to internal data packing constraints of the PG (and P331 probe).

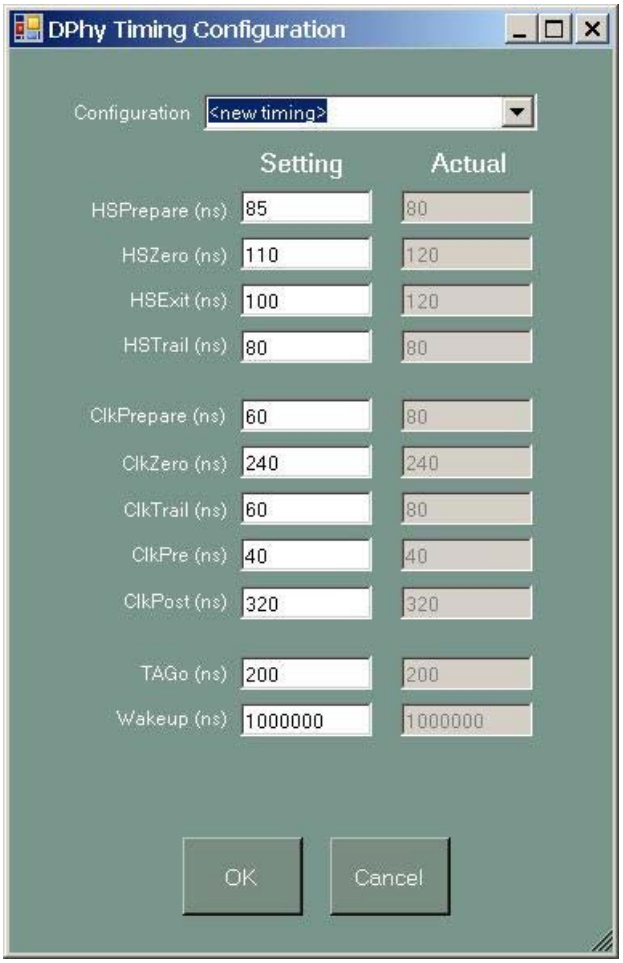

Figure 5 – DPhy Timing Configuration Dialog

The Configuration drop-down box always contains at least two options:

System Computed Timing: this option is the default setting of PGRemote. When selected this option disables custom editing of all parameters and requests that PGRemote automatically compute reasonable defaults. These defaults depend on and change with the current HS clock frequency. When this option is selected, the default settings for the current HS clock frequency are displayed in the read-only parameter boxes.

<new timing>: this option enables and fills in the left column parameter boxes with the current default settings. The user can then change values to create custom timing settings to use for all subsequent commands. Once a parameter is changed, the configuration name changes to <custom>. Please be aware that many DPhy timing parameters have a valid range dependent on HS clock frequency. Since custom values are do not change with clock frequency, some custom timing values will be valid only for a specific range of HS clock frequencies. No check is made by PGRemote for the validity of custom parameters with respect to the DPhy specification.

Other options in the Configuration drop-down box (if any) are those defined in the DPhy.Config text file (if present) located in the application directory.

Once a configuration has been selected, and values edited if desired, click OK to accept the new DPhy configuration settings and close the dialog. Otherwise, click Cancel to discard any changes and close the dialog.

3.5.1 DPhy.Config File Syntax

Default DPhy timing configurations available in the Configuration drop-down box of the DPhy Timing Configuration dialog are loaded from a user-editable text file when the dialog is displayed. This file is located in the PGRemote application directory and is called:

c:/Program Files/TMPC/PGRemote/DPhy.Config

It is probably helpful to start with an existing DPhy.Config file when creating a new one.

The following rules are used to parse the DPhy.Config file:

- Each line is either a comment line, blank line, or a parameter line.
- A comment line begins with "//" and is ignored by the parser.
- A blank line consists solely of spaces or tabs and is ignored by the parser.
- A parameter line has the format "<name> = <value>".
- Table 2 describes the parameter names that are supported. Names are case insensitive.
- Multiple timing configurations can be defined in the file. For each timing configuration in the file, the "Name" parameter must occur before any other associated parameters.
- Parameters not included default to the system default for the current HS clock

Table 1 – DPhy.Config File Parameters

Parameter	**Description**
Name	Defines a new timing configuration with the given name. Names may contain any alphanumeric character (no spaces, tabs, or equal signs)
HSPREPARE	Time to drive LP00 before starting HS burst (data lane)
HSZERO	Time to drive HS0 before sync sequence (data lane)
HSEXIT	Time to drive LP11 following HS burst (data lane)
HSTRAIL	Time to drive flipped differential state after last HS payload bit (data lane)
CLKPREPARE	Time to drive LP00 before starting HS clock (clock lane)
CLKZERO	Time to drive HS0 before sync sequence (clock lane)
CLKTRAIL	Time to drive HS0 after last HS clock (clock lane)
CLKPRE	Time to drive HS clock before any HS data lane transitions to HS mode (clock lane)
CLKPOST	Time to drive HS clock after last HS data lane has transitioned to LP mode (clock lane)
TAGO	Time to drive LP00 before releasing control during BTA.
WAKEUP	Time to drive Mark-1 state prior to Stop on exit from ULPS.

3.6 Video Mode

3.6.1 Overview

PGRemote operates the PG in a special mode called video mode when the user sends a video mode command. Video mode commands are DSI commands whose packet type is one of the pixel stream commands or any of the pixel data CSI commands (i.e. RGB, YUV, Raw, etc). Video mode commands are treated differently than non-video mode commands in that they cause PGRemote to build an entire DPhy packet stream representing one or more video frames, having the appropriate timing and structure according to the MIPI protocol standard. The video sequence can be played once or looped continuously to provide ongoing real-time video appropriate for testing of DSI displays or emulation of CSI camera chips. Note that all supported YCbCr formats currently implement only progressive frame construction.

All video mode commands accept a single DataFileName argument to specify a file containing the video frame data to send. To send a video sequence of more than one frame, this file has a special naming convention to allow PGRemote to derive the file names for subsequent frames in the sequence (see section 3.10.4.1).

Before sending a video mode command, additional parameters of video mode playback must be appropriately configured using the Frame Timing dialog (see section 3.6.3). These timing parameters, like other application settings, are saved and restored in an application configuration file when PGRemote is closed and restarted.

Once a video command has been defined and the timing configuration parameters have been set, sending the command initiates the sending of video data. The internals of how

video frames are constructed is described in section 3.6.7. Once video frame packets have been built, sent to PGAppDotNet, and downloaded to the PG3A hardware, video begins playing on the DPhy bus.

If the 'Play Mode' parameter is set to 'Continuous', the video frames are played and looped indefinitely, stopping only when the 'PG Abort' button at the bottom of the main window is pressed. When the PGRemote aborts the PG playback sequence, the PG completes a full loop of the sequence before returning to the LP11 quiescent state.

3.6.2 Command Insertion

If continuous looping is specified in the timing configuration, the PG will continue playing video packets indefinitely. To send another command, generally the "PG Abort" button should be sent to cleanly stop the PG from playing video and return to the quiescent LP11 state. Unless command insertion is enabled (see next paragraph), if a command is sent while video is playing , PGRemote will ask you whether you want to stop the PG to allow the sending of the command.

Command insertion is a special mode that allows non-video, non-read, non-macro commands to be sent during the vertical blanking of a video frame without stopping the PG or interrupting the video stream.[10] This function is enabled using the "Enable command insertion" option (Options menu). If this option is checked, a command sent while video is playing is automatically inserted into the video stream (during the first vertical blanking line). Note also that the insertion event is reflected on the Evt0 signal of the P331 back panel (see **Figure 2**), allowing for instrument trigger at the point of command insertion.

The inserted command may be sent in HSDT or LPDT mode and will inherit the LPFreq, HSFreq and LaneCnt parameters used for video playback.

3.6.3 Frame Timing Dialog

The "Timing Cfg…" button in the main window brings up a dialog to set your MIPI device frame timing parameters. The parameters in this window are used to configure video playback when one of the four DSI Pixel Stream commands are sent (corresponding to the four different RGB color packing formats) or when one of the numerous CSI Pixel Stream commands are sent (RGB, YUV, Raw, etc.)

Depending on the current MIPI protocol (DSI or CSI) the dialog will look a little different. **Figure 6** shows the DSI layout of the Frame Timing dialog:

[10] Note that command insertion has been extended to work in looping macros as well using the "Cmd Insertion Point" packet type (or CMD_INSERTION_POINT RPC command).

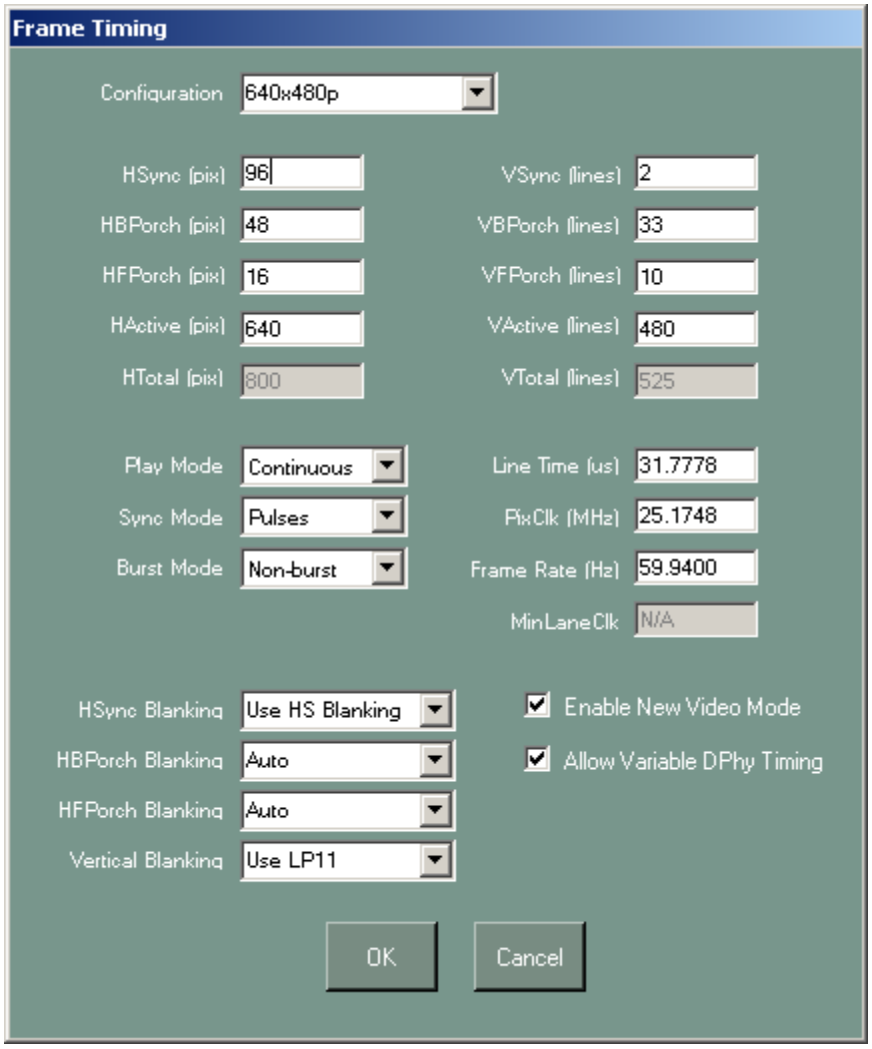

Figure 6 – Frame Timing Dialog (DSI)

- Controls in the top part of the window specify horizontal and vertical blanking component dimensions as well as the active dimensions of the frame. Read-only H and V totals are automatically computed as values are entered.

- Controls in the center-right of the window let you specify the pixel clock timing in four different ways, via: line time duration, pixel clock frequency, frame rate, or lane clock frequency (if enabled). Entering a value in one of these four controls automatically updates the other three.

- The MinLaneClk text box provides the lane clock frequency required to support the current timing configuration. This value will be dependent on the current timing settings, the current PktType setting in the main window (indicating the video format) and the LaneCnt. If the current PktType is not a video command, MinLanClk displays "N/A" and the text box is disabled. Otherwise, the text box is enabled, allowing the user to enter in an HS frequency. Note that if the Lane

Count is subsequently changed, the MinLaneClk frequency will change along with it (keeping the line time, pixel clock, and frame rate constant).

- The "Play Mode" drop-down box selects two playback options:
    - **Single Seq**: the frame(s) defined in the Pixel Stream command is(are) played only once. After playing, the PG returns to its quiescent state with LP11 asserted on all DPhy data lanes.
    - **Continuous**: the frame(s) defined in the Pixel Stream command are looped continuously. After the command is complete, the PG remains running, looping on the video frames
- The "HSync Mode" drop-down box selects the DSI HSync Mode:
    - **Events**: horizontal syncs in active lines of video frames are conveyed with only an HSync Start packet
    - **Pulses**: horizontal syncs in active lines of video frames are conveyed with an HSync Start packet, a Blanking packet, and an HSync End packet. In DSI, this option is only available in Non-Burst mode.
- The "Burst Mode" drop-down box selects the burst mode:
    - **Non-Burst**: non-burst mode assumes the DSI clock is just sufficient to transmit active lines with no extra bandwidth. If the clock is set such that there is extra bandwidth based on the current horizontal line parameters, it is consumed in the horizontal sync pulse width when HSync Mode is set to Pulses. Otherwise, the line time is shortened.
    - **Burst**: burst mode assumes the clock is set faster than necessary to transmit the video frame. Currently, a blanking packet is inserted in each line following active video to consume the excess bandwidth.

3.6.3.1 New Video Mode Implementation

More recently, the DSI video mode implementation has been significantly improved. The following summarizes the differences between old and new implementations:

Table 2 – Comparison Between Old and New Video Modes

Old Version	**New Version**
Software always tries to implement a blanking segment via LP11. Only if the period is too short is an HS blanking packet is used.	The user can identify for each blanking period whether it should be implemented using LP11 or HS blanking.(or Auto which implements LP11 if feasible and HS blanking otherwise).
When LP11 blanking is used, the blanking	Blanking periods are always implemented

period is always quantized to an even number of byte-clocks.	with the exact length as requested (or an error is reported that describes why the attempt failed). This includes blanking periods that end on non-integral byte-clock boundaries and lane boundaries other than zero. If blanking is implemented with LP11, a small HS blanking packet may be added at the end of the segment if necessary to implement non-zero lane boundaries.
Not all blanking periods can be implemented via HS blanking (at least one must use LP11 to allow quantization or an error is returned).	If desired, all blanking periods can be implemented via HS blanking.
EoTp is not supported in video mode	EoTp is supported in video mode (if enabled).
The HSync period represents the blanking time between HSS and HSE, not including either packet time. The HBackPorch includes the HSE packet time and the HFrontPorch includes the HSS packet time.	The HSync period includes the HSS packet time and HBackPorch includes the HSE packet time.
Interlaced formats are not supported.	Interlaced formats for all YCbCr modes are supported.
Frame sequences consisting of multiple frames cannot be stepped.	Frames sequences consisting of multiple frames can be stepped (i.e. played one-by-one) using an external event signal (see section 3.7.4)

Accordingly, additional controls have been added to the Frame Timing dialog to support the new implementation, which are enabled when the "Enable New Video Mode" check box is checked (when unchecked, the old video mode implementation is used and the new video mode controls are disabled). Here is a description of the new controls:

The major restriction of both DSI New Video Mode and CSI Video Mode implementations is that, generally, PGRemote requires that video lines have an integral number of bytes per lane. However, for DSI, if there are an even number of video lines in the frame (VTotal), PGRemote can pair lines together in the PG so that line lengths can be a multiple of ½ bytes per lane.[11]

[11] This requirement is a PGRemote requirement (not a MIPI requirement) and is due to how video is implemented in the PG. Video lines (or line pairs) are stored in the PG as individual blocks which, because of how video data is packed, requires quantization to lane byte boundaries.

Example:

For a video frame with an HTotal of 1516 pixels in packed RGB 6-6-6 format, there are 1516 * (18 bits/pixel) / (8 bits/byte) = 3411 bytes in a line. If you try to send this using 4 lanes, you will receive an error:

> "Error: <file> has invalid height or width for selected format"

This is because 3411 / 4 = 852.75 bytes / lane which is not an integral number of bytes. If you send this using 3 lanes, it will be successful, since 3411 / 3 = 1137, an integer. Finally, using 2 lanes, sending the video will be successful only if VTotal is even, since the number of bytes / lane is 3411 / 2 = 1705.5, an integral number of ½ bytes.

Note that if the video format is RGB888, all lane counts are feasible because there are 1516 * 3 = 4548 bytes / line, which is divisible byte 1, 2, 3, and 4.

Allow Variable DPhy Timing: to support LP11 blanking periods with half-byte-clock resolution the HSTrail and/or HSZero DPhy timing periods may need to be increased by a half-byte-clock. Normally, these timing periods (which are specified in the DPhy Timing Configuration dialog) are quantized to integral byte-clocks and do not change from segment to segment. To allow this adjustment as necessary for video mode implementation, the user must check this box or an error will be returned for video timing configurations that require it.

Turn clock off during LP blanking: selecting this option causes the clock to be turned off during LP blanking during video mode. Otherwise, the clock remains on throughout video playback.

Send "Top-Field-First": selecting this option causes the causes the top-field (i.e. lines 1, 3, 5…) of the image to be sent before the bottom-field (i.e. lines 2,4,6…) when sending interlaced video. Otherwise, the bottom-field is sent before the top-field.

HSync Blanking: this control allows the user to specify whether to implement the HSync blanking period via LP11 or an HS blanking packet. A third option is "Auto" which implements the blanking segments via LP11 unless it is too short in which case the segment is implemented with an HS blanking packet.

HBPorch Blanking: this control allows the user to specify whether to implement the HBackPorch blanking period via LP11 or an HS blanking packet. A third option is "Auto" which implements the blanking segments via LP11 unless it is too short in which case the segment is implemented with an HS blanking packet.

HFPorch Blanking this control allows the user to specify whether to implement the HFrontPorch blanking period via LP11 or an HS blanking packet. A third option is "Auto" which implements the blanking segments via LP11 unless it is too short in which case the segment is implemented with an HS blanking packet.

Vertical Blanking this control allows the user to specify whether to implement the blanking periods during Vertical blanking as LP11 or HS blanking packets. A third option is "Auto" which implements the blanking segments via LP11 unless it is too short in which case the segment is implemented with na HS blanking packet. Note that vertical blanking lines have one blanking period in "Event" mode and two blanking periods in "Pulse" mode (with the HSync End packet in between them).

3.6.4 DSI Interlaced Modes

The latest version of PGRemote supports interlace for all the YCbCr video packet types. A new parameter called 'Interlaced" is provided in the argument list for the command. Setting this value to '1' enables interlace for the implemented video mode when the command is sent.

Note that all of the vertical frame parameters in the frame timing dialog remain as *frame-based* settings. This means the number of lines given for VSync, VBPorch, and VFPorch are divided by 2 when implementing each field. Accordingly, the following restrictions are placed on the vertical frame parameters:

- VSync must be even
- VBPorch must be even
- VFPorch must be odd
- VActive must be even

Other requirements are that:

- 'Enable New Video Mode' must be checked
- If pulse mode is enabled, HSync blanking must be less than ½ line

To support the new Interlaced parameter using RPC commands, note that the PIXEL_STREAM calls now use arg2 to indicate whether or not interlaced mode is enabled.

3.6.5 CSI Frame Timing Dialog differences

Figure 7 shows the CSI layout of the Frame Timing dialog. Basically, the HSync Mode and Burst Mode controls have been removed and three new controls have been added:

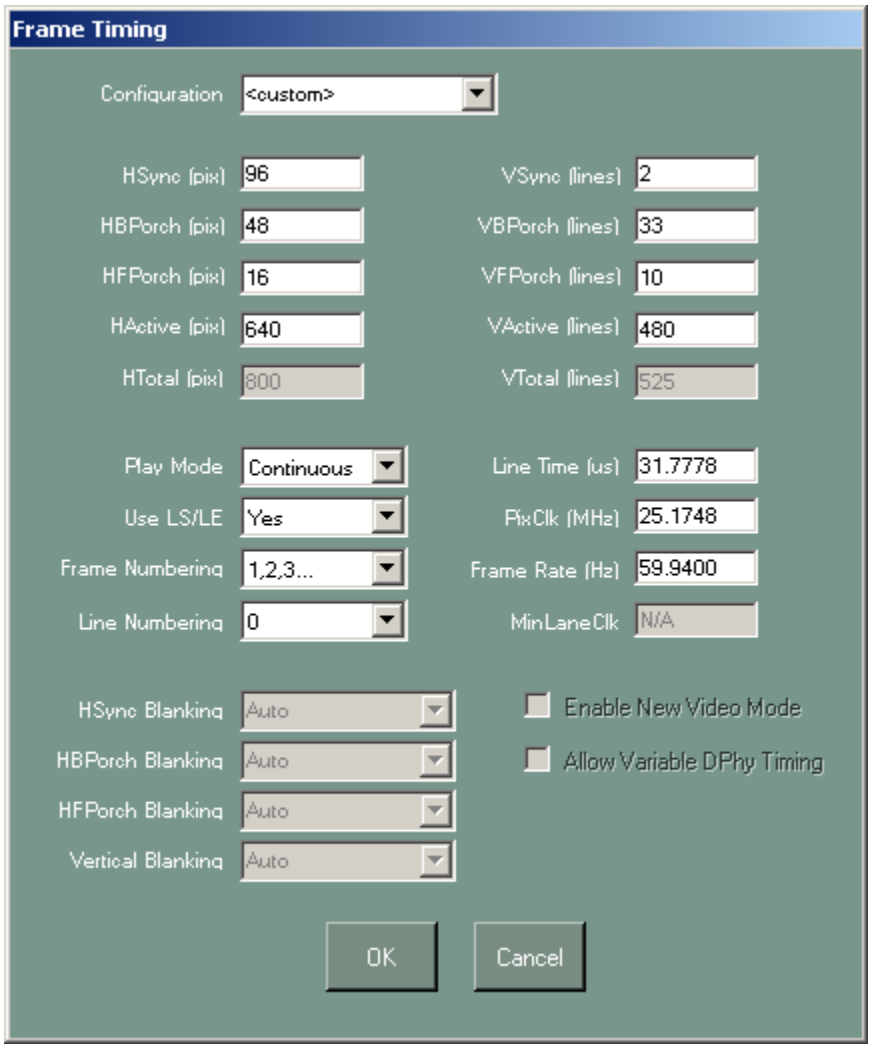

Figure 7 – Frame Timing Dialog (CSI)

- The "Use LS/LE" drop-down box selects Line Start/Line End packet usage:
    - **Yes**: Line Start and Line End packets are inserted into the video stream and bracket each active video packet.
    - **No**: Line Start and Line End packets are not inserted into the video stream.

- The "Frame Numbering" drop-down box selects Frame Numbering usage:
    - **0**: the frame number parameter in the Frame Start and Frame End packets is set to zero.
    - **1,2,3**…: the frame number parameter in the Frame Start and Frame End packets is set, starting with 1 and incrementing each frame.

- The "Line Numbering" drop-down box selects Line Numbering usage:

- **0**: the line number parameter in the Line Start and Line End packets is set to zero.
- **1,2,3…**: the line number parameter in the Line Start and Line End packets is set, starting with 1 and incrementing each line.

3.6.6 Timing.Config File Syntax

Default timing configurations available in the drop-down box of the Timing Config dialog are loaded from a user-editable text file when the dialog is displayed. This file is located in the PGRemote application directory and is called:

c:/Program Files/TMPC/PGRemote/Timing.Config

It is probably helpful to start with an existing Timing.Config file when creating a new one.

The following rules are used to parse the Timing.Config file:

- Each line is either a comment line, blank line, or a parameter line.
- A comment line begins with "//" and is ignored by the parser.
- A blank line consists solely of spaces or tabs and is ignored by the parser.
- A parameter line has the format "<name> = <value>".
- Table 2 describes the parameter names that are supported. Names are case insensitive.
- Multiple timing configurations can be defined in the file. For each timing configuration in the file, the "Name" parameter must occur before any other associated parameters.
- Parameters not included for a timing configuration default to 0.

Table 3 – Timing.Config File Parameters

Parameter	**Description**
Name	Defines a new timing configuration with the given name. Names may contain any alphanumeric character (no spaces, tabs, or equal signs)
HSync	Duration of horizontal sync in pixels
HBPorch	Duration of horizontal back porch in pixels
HFPorch	Duration of horizontal front porch in pixels
HActive	Number of active pixels in line
VSync	Duration of vertical sync in lines
VBPorch	Duration of vertical back porch in lines
VFPorch	Duration of vertical front porch in lines
VActive	Number of active lines
LineTime	Duration of line in microseconds. Only one of LineTime, PixClk, or FrameRate is required in a timing configuration.
PixClk	Pixel clock in MHz. Only one of LineTime, PixClk, or FrameRate is required in a timing configuration.
FrameRate	Frame rate in Hz. Only one of LineTime, PixClk, or FrameRate is required in a timing configuration.
PlayMode	Defines how the PG will play video frame(s). Set to "0" for single-sequence or "1" for continuous playback.
HSyncMode	Defines how HSync will be output. Set to "0" for events and "1" for pulses.
BurstMode	Set to "0" for non-burst mode output and "1" for burst mode output.
FrameIncr	Set to "0" to set the frame number parameter in Frame Start and Frame End packets to 0. Set to "1" to use an incrementing sequence beginning with 1.
LineIncr	Set to "0" to set the line number parameter in Line Start and Line End packets to 0. Set to "1" to use an incrementing sequence beginning with 1.
UseLSLE	Set to "0" to disable and "1" to enable Line Start and Line End packet insertion.
Use_New_Video_Mode	Set to "0" to use the old DS I video mode implementation and "1" to enable the new video mode implementation.
Allow_Variable_Dphy_Timing	Set to "0" to require lines to end on a byte-clock boundary and "1" to support lines ending on a half-clock boundary. This is accomplished through adjusting the HSZero and/or HSTrail timing if necessary on burst entry/exit.[12]
Turn_Clock_Off_During_LP_B	Set to "0" to keep the HS clock running throughout video

[12] Applies only to the new video mode implementation for DSI (i.e. when Use_New_Video_Mode is non-zero).

lanking	output and "1" to allow the clock to turn off and on during LP11 blanking times (length permitting). [12]
Top_Field_First	Set to "0" for the bottom field to output before the top field in DSI video interlaced modes. Set to "1" for the top field to be output first.
HSync_Blanking_Mode	Set to "0" or "Auto_Blank_Mode" to implement HSync blanking via automatic determination (LP11 if length permits, HS blanking otherwise). Set to "1" or "HS_Blank_Mode" for HS blanking and "2" or "LP11_Blank_Mode" for LP11 blanking. [12]
HBPorch_Blanking_Mode	Set to "0" or "Auto_Blank_Mode" to implement HSync blanking via automatic determination (LP11 if length permits, HS blanking otherwise). Set to "1" or "HS_Blank_Mode" for HS blanking and "2" or "LP11_Blank_Mode" for LP11 blanking. [12]
HFPorch_Blanking_Mode	Set to "0" or "Auto_Blank_Mode" to implement HSync blanking via automatic determination (LP11 if length permits, HS blanking otherwise). Set to "1" or "HS_Blank_Mode" for HS blanking and "2" or "LP11_Blank_Mode" for LP11 blanking. [12]
Vertical_Blanking_Mode	Set to "0" or "Auto_Blank_Mode" to implement HSync blanking via automatic determination (LP11 if length permits, HS blanking otherwise). Set to "1" or "HS_Blank_Mode" for HS blanking and "2" or "LP11_Blank_Mode" for LP11 blanking. [12]

3.6.7 Video Frame Construction Details

In contrast to the relatively simple non-video mode commands, video-mode commands define numerous blocks and have a more complicated sequence. A PG block is built for each unique line segment, such as the start and end of active lines that have horizontal sync and blanking, active line segments comprised of video packets, and several segment definitions to build vertical blanking lines. They are all put together in the sequence to play out one or more video frames.

While it is not generally recommended to interact with the PGAppDotNet server while PGRemote is connected, in this case it is constructive to send a video command and then look at the PGAppDotNet programming, i.e. the blocks and sequence that are created.

DSI frames consist of vertical blanking lines and active lines. Structurally, lines consist of various segments defined by the DSI specification: horizontal sync (HSA), horizontal front-porch (HFP), horizontal back-porch (HBP), and active video. The duration of each of these segments (in pixels) is specified in the timing configuration.

In PGRemote, a sync or blanking segment is implemented by HS blanking packet or a round-trip transition to LP11 for the required duration. Blanking segments long enough to incur the overhead of the round-trip transition to LP11 use this method. Otherwise, the

interval is implemented with a blanking packet. Of course, even blanking packets have overhead, and so segments must be at least long enough to accommodate the minimum length blanking packet.

While the desired timing characteristics of a video line is defined in the frame timing dialog, actual timings of may slightly differ. This is because of the complexity of constructing arbitrary packet timing given bit packing and vector alignment of data in the PG, as well as the variable overhead associated with DSI/DPhy byte quantization, lane multiplexing, and SOT/EOT signaling requirements.

The following summarizes notes for building DSI video frames and lines:

- Frames begin with the start of vertical blanking and end with the last active line in the frame.
- Lines have the following segment order: HSA, HBP, active, HFP.
- Vertical blanking lines merge HBP and active line segments in to HFP.
- Blanking segment timing takes into account all packet overhead and/or SOT/EOT signaling overhead
- Any line time adjustments are made to the last blanking segment (HSA, HBP, HFP) that is implemented with a transition to LP11 (i.e. not a blanking packet). This is because fine-time adjustments can't be made in a blanking packet. If no such segment exists, video frame construction fails.
- If HSync Event Mode is true, the HSA segment begins with the HSync Start packet and the total duration is the combination of HSA and HBP. The HBP segment is not then implemented.
- Active packet overhead is subsumed into HBP.

CSI video frame structure is simplified compared to DSI. The following summarizes notes for building CSI video frames and lines:

- Frames begin with the second line of vertical blanking and end with the first line of vertical blanking in the frame (which contains the Frame End packet at the beginning).
- Lines have two segments: Blanking and Active (with optional Line Start and Line End packets)
- Blanking segments have total duration equal to the HSA, HBP and HFP segments in the timing configuration.

3.6.7.1 Video File Sizes for P331

In non-burst mode, DSI, the amount of PG memory needed to accommodate a particular video format is approximately as follows:

Number of vectors = HTotal * (VActive + 7) * (BitsPerPixel) / (4*LaneCount)

In non-burst mode, CSI, the amount of PG memory needed to accommodate a particular video format is approximately as follows:

Number of vectors = HTotal * (VActive + 3) * (BitsPerPixel) / (4*LaneCount)

Conversely, the number of frames that will fit into a PG3A is roughly:

32,000,000 / (number of vectors).

3.7 Other Special Modes and Behavior

3.7.1 Clock Lane Behavior

The DPhy clock lane originally was intended to turn on with the first MIPI command and remain on for all subsequent commands (with the exception of ULPS which came later). Requests to provide on/off behavior subsequently led to the menu option "Turn clock on/off with each command". This option applied only to non-video commands and treated macros as a single command.

To allow the user for greater flexibility in controlling the clock lane (on/off), two new PktType commands have been introduced: "Clock On" and "Clock Off". These commands perform MIPI conformant transitions between LP11 and HS bits toggling on the clock lane. These commands are generally intended for use in macros, to allow for control over the clock during component macro commands.

The following then summarizes behavior and assumptions about the clock lane state (these do not apply when in ULPS mode):

- When "Turn clock on/off with each command" is disabled, the assumed state of the clock when any command is sent is ON (HS toggling), and the state of the clock when commands complete is ON.
- When "Turn clock on/off with each command" is enabled, the assumed state of the clock when any command is sent is OFF (LP11), and the state of the clock when commands complete is OFF.
- If the clock is not in the assumed state when a command is sent, it is automatically turned ON or OFF as appropriate before sending the command, with one exception. The exception is: if the command being sent is a macro, and the first component command of the macro turns the clock on or off, no automatic action is taken. This avoids, for example, the clock to be automatically turned on only to be turned off again by the first component command of a macro.
- Within a macro, if the clock lane is already ON when a "Clock On" command occurs, no action is taken
- Within a macro, if the clock lane is already OFF when a "Clock Off" command occurs in a macro, no action is taken.
- While the restart command will allow replaying a macro with clock on/off commands, the clock lane may not behave as intended if the initial clock state is different upon restart as it was when the command was initially sent.

A couple implications of the above behavior are:

- The only way to send an LP command with the clock turned OFF is through a macro.
- A macro that ends with a clock on/off command does not guarantee the clock lane state after it completes. This is determined by the option "Turn clock on/off with each command".

3.7.2 ULPS

Two commands in the PktType list-box are available for entering and exiting ULPS respectively: "Enter Ultra Low-Power State" and "Exit Ultra Low-Power State". While these commands cannot be added to macros, they can be assigned to buttons and are supported via RPC commands as well.

When ULPS is entered, all lanes including the clock lane are put in ULPS according to the DPhy protocol. When lanes are in ULP state, the "Lane Clk" and "Lane Data" status in the bottom right of the main window will indicate "ULPS".

Like continuous video mode, the PG continues to run while the ULP state is held and when the PG is stopped, the ULP state is exited. There are three ways to exit ULP state:

1) Via the "Exit Ultra Low-Power State" command
2) Via the "PG Abort" button
3) By sending a MIPI command

In each case, all lanes follow the protocol to exit ULP state. After exiting ULPS, the HS clock lane may start if the "Turn clock on/off each command" option is not selected. Finally, in case 3, the MIPI command is then sent.

3.7.3 External Event Input Triggering

The option "Run PG on external event" allows users to configure the PG to wait for an external trigger event before sending any command (including video mode frames) When this option is enabled, commands that are sent to the PG are prepared and sent as usual to the PG but instead of the data being sent out, the PG is "Armed" and waits for a rising edge (logic level '0' to '1') on event line 0 of the P300 Inputs probe connected to the PG.

Once this edge is seen, the command is output. The latency of the event signal through the PG and P331 probe is clock frequency dependent, but should be on the order of microseconds.

Independent of this option, the external event input can also be used as a method to pause output (held in a specified static LP state) while playing a macro. The command to use for this function is WAIT_EXT_EVENT (PktType = "Wait Ext Event").

3.7.4 Video Sequence "Stepping" Using External Events

In addition, when the option "Run PG on external event" is used in conjunction with the "New Video Mode" DSI implementation, a sequence of video frames can be looped individually, one at a time. The external event is used to advance to the next video frame in the sequence.

Conditions to enable this mode:

1. Run PG on external event" option is selected (Options menu)
2. New Video Mode is checked in the Timing Configuration
3. Continuous video mode is selected in the Timing Configuration
4. A video command containing multiple video frames is sent (i.e. FrameCount argument is greater than 1)

When the video command is sent, the PG waits for an initial event before playing the first video frame (see previous section). Then, once the event line 0 of the P300 Inputs probe connected to the PG sees a rising edge the first video frame is played indefinitely. Then, when a second rising edge event occurs, the second video frame is played. Etc. After the last frame in the sequence has played, the sequence starts over with the first frame again.

3.7.5 Causing Packet Errors

The PGRemote GUI supports the ability to modify a packet's ECC and CRC fields to cause receive errors in packets. Normally, when a packet type is selected for a command in the main window and its parameters are filled in, read-only fields are displayed showing the DataID and ECC for short packets and, additionally, the WordCount and CRC for long packets. Checkboxes next to the ECC and CRC fields allow user modification of these fields to cause a receive error in the packet.

To change the ECC or CRC, simply check the associated checkbox, making the field value editable. Unchecking the checkbox, recomputes the correct ECC or CRC value and restores the field back to its original read-only state.

Note that this capability applies to all single commands and commands within macros as well. The File command does not break-out ECC and CRC fields (as it can contain multiple packets) so this capability is not applicable to File commands. Similarly, with video mode, there is no place for the user set the ECC or CRC of any of the video frame packets.

When applied to the WriteMemory commands, if the command is partitioned into multiple WriteMemory commands, the user-set ECC and/or CRC applies only to the first WriteMemory command. Remaining WriteMemory commands are always built with their correct ECC and CRC.

An additional mechanism for introducing packet errors is described in the description of RPC. In addition to allowing the ECC and CRC to be set, the SEND_IMPAIRED_MIPI_CMD RPC call allows any byte in a packet to be XORed with

an error mask before it is sent. Please see the description of this command for further information.

3.7.6 Script Recording

Script recording is a mode in which MIPI commands sent to the DUT are also written as RPC commands to a designated script text file. All commands, including macros and video mode commands are recorded. After recording, the script file can be sent as any other script file or used as a template for creating additional script files.

In addition, the current configuration state can be written to the script file. This allows the script file to initialize PGRemote to a particular configuration or be sent to the Moving Pixel Company to document your configuration when requesting assistance.

To enable script recording, select "Start Recording" from the Script menu. When this option is selected, a browse dialog is opened for you to designate the text script file name to use for recording.

Once script recording is enabled, the status bar displays the text "Recording" in a new pane. The script file is initialized with a header containing information about the current software and hardware configuration. Here is an example header:

```
// Script file created 4/3/2012 10:24:44.9838135

// PGRemote version: 1.2.2.0
// PGAppDotNet software version: 2.00.27
// PG module serial number(s): 1063
// PG module firmware revision(s): 1.9
// PG options:   CSI, DSI, DDR, DIGRF
// Probe type on module 1, connector A: P331
// Probe version is: 1.1
```

Any MIPI command sent (without error) during recording also causes an RPC command to be appended to the script file. In addition, selecting the menu option "Write Current State" from the Script menu causes the current state to be written to the script file. To end script recording, select "End Recording" from the Script menu.

3.7.7 Generic Standard

The Generic standard option in the Standard menu is not a MIPI standard and is not intended to be used with MIPI devices. Rather it allows the P331 probe to be used as a generic high-speed serial probe, outputting data specified by the standard's only command which is the File Command.

The File Command syntax is identical to that used with CSI/DSI standards, allowing the use of either ASCII or binary file format. The ASCII file format has the following syntax:

- A command line begins with "//" and is ignored by the parser.
- A blank line consists solely of spaces or tabs and is ignored by the parser.
- The first non-comment, non-blank line should be "#ASCII".

- Subsequent lines define hex data bytes separated by white-space or commas.
- Any number of hex data bytes are allowed per line.

Example:

```
// This is an example ASCII file for the File Command
#ASCII
18 04 00 4f
87 4d 22 8c
```

If the parser cannot find the #ASCII token in the specified file, it is assumed to be a binary file. In this case, entire contents of the file are considered to comprise a data packet or series of packets to associate with the command name.

Though the file format is the same as for the CSI/DSI standards, the interpretation of bytes for output is different. In the generic standard, data bytes are interpreted as a stream of bits, ls-bit first. Then, bits are interleaved across lanes depending on the current lane count. Finally, if the number of bits is not evenly divisible by eight times the number of lanes, the last bit is replicated so that each lane has an even number of bytes.

When the command is sent, bits are output serially on lanes at two times HS clock frequency (bits are output on both clock edges). At the end of the command, the last bit of each lane will be held on the outputs until the next command is sent.

Example:

```
#ASCII
03 f0 80 91
```

If LaneCnt == 1, the following pattern will be output on lane 0:

```
1 1 0 0 0 0 0 0…
0 0 0 0 1 1 1 1…
0 0 0 0 0 0 0 1…
1 0 0 0 1 0 0 1…
```

If LaneCnt == 4, the pattern will be

```
1 0 0 1 0 0 1 1… on lane 1
1 0 0 1 0 0 0 0… on lane 2
0 0 0 1 0 0 0 0… on lane 3
0 0 0 1 0 1 0 1… on lane 4
```

3.7.8 Dual-video Synchronization

Special support has been added to PGRemote to support the approximate synchronization of video from two full, independent systems (PG3A, P331, PGAppDotNet, and PGRemote). Note this option requires at least firmware version 1.7 on the P331 probes. To make use of this support, in both copies of PGRemote, select the "Configure for dual-synchronized video" menu in the Options menu. This brings up a dialog with two fields:

- **Enable Synchronization** – check to enable support for synchronization

- **Delay after sync** – set this floating point field to the number of microseconds to delay video start after synchronization. This allows control over the relative start times between the video output from each system.

In addition to enabling synchronization, both copies of PGRemote should have the "Run PG On Event" and the "Use External Clock Input On P331 As 10 MHz Reference" options checked. With these options, PGRemote will build in the following behavior when video is sent:

> The TRIGOUT signal on each PG3A module will pulse high initially then immediately wait for an external high signal on the Event0 line of its Inputs probe. When the event occurs, both P331 probes are reset to an approximate known state, causing subsequent data to be output in near synchrony (less than 1 us).

As indicated, this mode requires two full PG systems (PG and P331) as well as two PGAppDotNet and PGRemote instances running. To bring up two versions of PGAppDotNet, you can just start them in sequence. The first will default to port 1025 and the second will default to port 2798. Next bring up two versions of PGRemote. Then connect to the servers making sure you specify which server port to use with the syntax <hostName>:<portNum> (see section 3.3 for more information).

For ease of use, the 10 MHz reference clock output (REFOUT) from the master PG can be tee'd and connected to both P331 external clock inputs. In addition, the TRIGOUT output from the master PG can be tee'd and connected to event 0 of both PG's Inputs probe.

Once configured, video is sent in both instances of PGRemote, first in the slave PGRemote, then in the master PGRemote. The slave will download and initialize but wait in the LP11 state for the master to start. When the master starts, both will start outputting video, nearly synchronized.

3.8 Customizing GUI Colors

In the PGRemoteForP338 application, the user can configure the colors of various elements of the GUI (e.g. the background and foreground colors of buttons, forms, menus, etc). This is achieved using the Color Options dialog, brought up via the Options menu by selecting the "Set Colors" menu item.

The Options dialog (see Figure 8) consists of a column of buttons on the left showing the current color settings of GUI controls. In the center of the dialog is a color square showing all the colors associated with a given luminance. Clicking and dragging the mouse inside of the color square will select the color of the currently selected control.

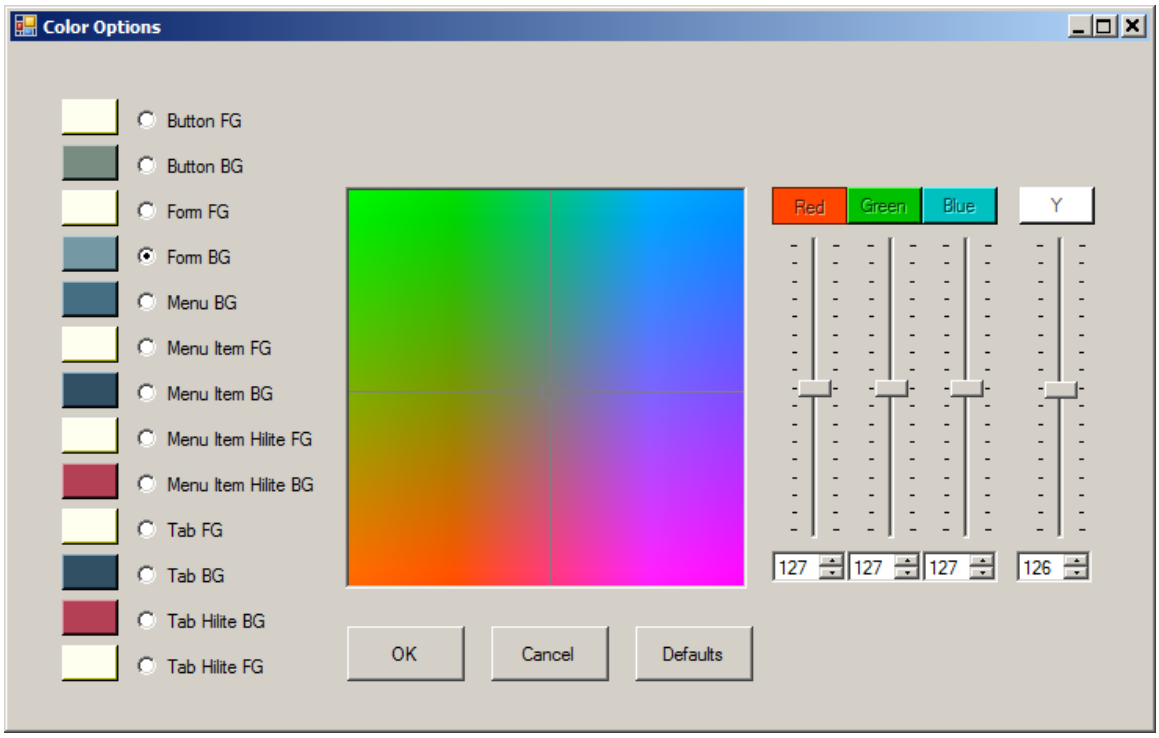

Figure 8 – Color Options Dialog

The luminance can be changed by adjusting the slider labeled 'Y'. Alternately, you can change the R, G, B color components using the sliders on the right of the color square. Yet another way to change the color is to click on a control button color (far right column). This loads the selected color as the current color. For example, to set the Button foreground the same as the Form foreground, click the Button FG radio button and click the color button next to the Form FG button.

As the control color changes, the main window colors will change in real time, so you can see how colors interact. To restore the default colors, click on the Defaults button. When you are done changing colors, to accept the new colors, click the OK button. To discard your color changes, click the Cancel button. Color settings are saved on application exit and restored when the application is restarted.

3.9 Menus

This section outlines the menu commands available in PGRemote:

- **File**:
  - **Load –** loads a previously saved command configuration file, overwriting any current commands and button assignments.
  - **Save <fn> --** saves the current command definitions and button assignments from both the CSI and DSI protocol sets to the current file <fn>. Note that the command configuration file does not store application

settings, which includes PG configuration and Timing configuration settings.
    - **Save <fn> As… --** same as the **Save <fn>** command above except a dialog appears to browse and enter a new configuration file name.
    - **<recent files>** a list of the four most recent configuration files. Selecting one of these file names loads the file.
    - **Clear Recent File List –** clears the most recent file list
    - **Send Cmd To File** – send the current command as ASCII bytes to a file (allowing it to be modified and sent using the File Command packet type). Note: this option is only enabled for non-video, non-macro, non-DPhy commands when connected to a licensed PG.
    - **Exit** – exits the application
- **Connect**:
    - **PG Connect…** – brings up the PG connection dialog (see section 3.3)
    - **PG Disconnect** – disconnects from the current PGAppDotNet server, leaving PGRemote in offline mode.
    - **Enable RPC** – when checked, enables PGRemote to accept and process incoming RPC requests (see section 4.1).
- **Standard**:
    - **DSI** – selects the DSI command set and button assignments
    - **CSI** – selects the CSI command set and button assignments
    - **Generic** – selects the Generic command set and button assignments.
- **Options**:
    - **Turn clock on/off each command** – this option is enabled only for the P331 probe. If checked, every time a command is sent, the clock lane begins in LP11 mode and transitions to HS mode (according to the DPhy specification) before the command is sent. After the command is sent, the clock lane transitions back to LP11 mode. If this option is not checked, the clock lane will be left oscillating in HS mode for continuous clock operation.
    - **Display command on button hover** – if checked, this option causes the command assigned to a button to be displayed in the left pane temporarily when the mouse hovers over the button. The left pane reverts to its current settings when the mouse leaves the button.
    - **Enable command insertion** –if checked, a non-video, non-macro, non-read command sent while a video mode is looping is automatically inserted (once) into the vertical blanking interval without stopping video playback. If unchecked, sending any command while a video mode command is looping causes PGRemote to ask whether you wish to stop video playback to send the command. This function has been extended to work with a looping macro containing a command insertion point.
    - **Enable EoT packets** – this option, when checked, causes EoT packets to automatically be appended to non-video HS bursts.
    - **Disable Command Timeout** – this option, when checked, disables the normal timeout for non-video commands. This timeout, which is set to about one second, causes the PG to abort output (exactly as if the user

pressed the Abort button). This option is particularly useful for macros that include BTA or BTA_WAIT commands, which can cause arbitrary delays based on the testing environment and the behavior of the device under test.

- **Loop non-video commands –** this option causes any non-video command or macro that is sent to be looped indefinitely. For HS commands, a short period of LP11 occurs between iterations. Similar to continuous video mode, looping can be stopped using the PGAbort button. Generally, this option is used for low-level debugging and so is not saved/restored with other application configuration settings.
- **Run PG on external event –** this option, when checked, configures the PG to wait for an external event on its P300 Inputs probe before continuing to output the data for a command. When event line 0 is a logic '0', the PG waits until it goes to '1' before continuing (see section 3.7.3 for more information). Note that the external event can also be used to control macro output using the "Wait Ext Event" packet type.
- **Use external clock input on P331 as 10 MHz reference** – this option allows the user to connect an external 10 MHz clock source to the P331 "Opt Clk In" input to be used as the reference clock for the data output frequency. Note this function requires at least P331 firmware version 1.7.
- **Don't force LP11 after command** – this legacy option is always disabled (since the discontinuation of support for the P375 probe for MIPI).
- **Send single packet per HS burst** – this option is currently enabled only for the CSI standard and affects all composite commands, including macros, multiple packets in a File Command (if legally parsible), and video mode commands. Normally, these commands send consecutive HS packets in a single HS burst. However, when this option is checked, each HS packet is sent in its own HS burst.
- **Use only LP11 for video blanking** – this option is enabled when "Send single packet per HS burst" option is checked.[13] In video frames, blanking can be implemented with either the LP11 state or a HS blanking packets. The default PGRemote implementation chooses whichever method is convenient (often depending on how long the time to assert blanking is). When this option is checked, video blanking is built solely using the LP11 state.
- **Configure DPhy Timing…** – selecting this menu option brings up the DPhy Timing Configuration dialog, allowing the user to enter custom DPhy timing parameters, most pertaining to the timing of clock and data transitions from LP to HS and HS to LP mode.
- **Configure WriteMemory commands…** – this option brings up the ConfigWriteMemory dialog for configuring whether and how a long DCS WriteMemory command is partitioned into multiple WriteMemory commands (see section 3.10.4.4 for more information).

[13] While the two options "Send single packet per HS burst" and "Use only LP11 for horizontal blanking" are not functionally related, they are implemented using a common subroutine. Thus, the second option is only available if the first option is used.

  - **Configure for dual synchronized video…** – this option brings up a dialog for synchronizing the video for two full, independent systems (P331, PG3A, PGAppDotNet, and PGRemote).
  - **Set Colors…** – this option is only available on PGRemoteForP338. When selected, the Color Options dialog is shown, allowing the user to change colors of various elements of the GUI.
- **Script**
  - **Start Recording** – begins a script recording of MIPI commands sent to the DUT. See section 3.7.6 for more details.
  - **End Recording** – ends script recording.
  - **Write Current State** -- writes all configurable state as RPC commands to the current script recording file (this options is only enabled during script recording).
- **About**
  - **Help** – displays this manual as a help file
  - **About** – brings up a summary window displaying the current software version. This dialog also includes a summary of release notes of changes/fixes for each version.

3.10 Commands

3.10.1 Defining a New Command

Commands are defined using the left pane controls of the main window and are grouped according to MIPI protocol. Thus, before defining a command, the user should select the desired protocol via the Standard menu, currently either CSI or DSI. Then, to define a new command, perform the following steps:

1) Select the command type via the PktType ComboBox.
2) If the command type is a DCS command from the DSI command set (i.e. DCS Short Write, DCS Long Write, or DCS Read Request), the DCSCmd ComboBox will be enabled. Select the DCS command type from this control.
3) Depending on the command selected, argument fields will be labeled and enabled for the user to enter values. Certain arguments are pre-filled such as the virtual channel, DT Mode, and BTA. The user may change these pre-filled values if desired.
4) Commands that support sending either as a LP or HS command initially have their DTMode argument set to "Default", which means the command will be sent using the current default DTMode setting (DT Mode control in the lower-left of the main window). Alternatively, the user can specify the DTMode for certain commands by changing the DTMode argument to either LPDT or HSDT.
5) Setting BTA to "Yes" requests that a bus-turn-around sequence follow the sending of the command. This is usually used for read commands to allow the device under test to respond with data. Note that the data response of the DUT is not monitored by the PG, only the return BTA sequence, so the PG knows when it can resume transmitting (See section 3.10.4.8 for more details).

6) Numeric fields are assumed to be decimal unless appended with 'h' (hexadecimal) or 'b' (binary). For example "67h" = 103 and "1101b" = 13.
7) Certain derived fields of the command will also be displayed in read-only text boxes (e.g. DataID, ECC, CheckSum). These fields cannot be modified directly by the user but are computed according to the MIPI specification based on the command arguments.[14] Note that video commands do not fill in these fields because they do not represent a single packet.
8) Once the user is happy with the command definition, he may choose to save the command so that it can be assigned to a command button and saved as a named command in a configuration file. To do this, the user should type a unique command name in the CmdName ComboBox and click on the Save button.
9) After the command is saved, the label in the CmdName box changes to "untitled" to prevent confusion when the user then subsequently changes argument values. In this case, he is not changing the saved command definition but is beginning a new command definition.

3.10.2 Assigning a Command to a Button

Once a command has been defined, the user can right-click on a button to bring up a context menu. This menu allows the user to select a command to assign to the button. Note that buttons maintain separate DSI and CSI command sets, selected when the protocol is changed using the Standard menu.

After a command is assigned to a button, if the "Display command on button hover" option is checked (see section 3.7), the user can see the command definition merely by hovering the mouse over the button. In this case, the command fields in the left pane of the main window will be filled in with the command definition. If this option is not checked, the user must select the command for editing to view its arguments in the left pane of the main window (see next section).

3.10.3 Editing an Existing Command

An existing command definition can be edited as follows:

1) Load the command field values into the edit controls either by right-clicking on a button having a command assignment and selecting "Edit" or selecting the command name from the CmdName drop-down control.
2) Edit the command fields
3) Click the Save button. A confirmation message will ask whether it is OK to overwrite the existing command definition. Click OK to do so.

3.10.4 Special Command Types

[14] To modify read-only fields, the user can use the "Send Cmd To File" menu option (File menu). This outputs the command bytes to a text file that can subsequently be modified and resent using the File command.

3.10.4.1 Video Command Definition

The four DSI video commands (i.e. Pixel Stream commands) and all of the CSI video commands all take a data file name and a frame count as arguments. The data file name should reference a file with one of the following characteristics:

- A file having the extension ".bmp". This file should be a 24-bit RGB BMP file having the exact width and height required HActive and VActive fields of the current timing configuration (see section 3.6.3). In this case, the BMP image is read in and converted to the correct format for the associated video command.[15]

- A file having any other extension. This file should be a binary file having at least the number of bytes required by the current timing configuration. That is, HActive * VActive * BytesPerPixel. In this case, bytes are read in and treated as video data in the correct format for the associated video command.

To play a sequence of multiple frames, the data file name should reference the first frame of the sequence and have the following format: "<name><index>.<ext>" where <name> is an arbitrary file name, <index> is an integer and <ext> is the file extension. PGRemote will derive the file names of subsequent frames automatically by incrementing <index>.

For example: if "c:\frame1.bmp" is specified as the data file name and the frame count is set to four, PGRemote will look for the files "c:\frame1.bmp", "c:\frame2.bmp", "c:\frame3.bmp", and "c:\frame4.bmp".

3.10.4.2 Variable Argument Commands

Certain DSI commands allow the user to specify a variable number of parameters through an argument text box named "Params[]", specifically the following commands:

1. Generic Short Write Command
2. Generic Read Request
3. DCS Short Write, Custom DCS Command
4. DCS Read Request, Custom DCS Command
5. Custom Command

The differences between these commands are minimal. The Custom Command allows the user to specify an arbitrary DataID byte. The other commands implicitly fill in the Data Type and allow the user to specify a Virtual Channel argument. The DCS commands also allow the user to specify the DCS command byte. All allow a variable number of arbitrary arguments, computing the packet length (for long packets), ECC, and Checksum (for long packets).

The actual packet constructed may have either long or short packet formatting, depending on the number of values parsed from the Params[] argument string. For packet types (1),

[15] Note for CSI Raw<N> video commands, image data is converted to YUV format and the luminence channel data is used with the appropriate precision.

(2) or (5), if Params[] contains 0, 1, or 2 bytes, then short packets will be constructed. For packet types (3) or (4), if Params[] contains 0 or 1 byte, then short packets will be constructed (prepended with the DCSCmd byte). Otherwise, long packets will be constructed and you will see additional read-only fields for the packet length and checksum appear.

The Params[] argument string consists of byte values separated by white-space or commas. As with other argument fields, values are assumed to be decimal unless appended with 'h' (hexadecimal) or 'b' (binary). For example, "30 41h 10101b" is a legal parameter string.

3.10.4.3 Long File Format Commands

Two DSI packet types, the Generic Long Write and DCS Long Write formats, have a DataFileName argument. For these commands, the file will generally be treated simply as a binary file. However, the ASCII file format described in the section about the File Command can also now be used with these commands. Please see this section for more information.

Regardless of the file format used, for the Generic Long Write and the Write LUT commands, the entire file is sent in the long packet. The WriteMemory commands have special support in PGRemote because these commands are often used to send video data to a frame buffer on the DUT. The next section describes this support.

3.10.4.4 DCS WriteMemory Commands

For the Write Memory Start and Write Memory Continue commands, the WriteLen argument specifies the number of bytes to send and the FileOffset argument specifies the starting byte offset in the file or frame buffer.

IMPORTANT: the parameter name FileOffset is misleading when using BMP files and the video format is not RGB888. The parameter would better be called FrameOffset. The reason is that this parameter is used *after* the file has been imported and converted to the output video format.

For example, for a 100 x 100 image in RGB666 (18-bits per pixel), an offset of 1800 references pixel 800 (= 1800 / (18/8)), which is the first pixel of line 8. In RGB888, an offset of 1800 references pixel 600, the first pixel of line 6. The line length in RGB666 is 100 * (18/8) = 225 and, in RGB888, is 100 * 3 = 300

PGRemote has additional features that enhance the behavior of sending WriteMemory commands:

- **BMP File Decoding –** If a file has extension ".bmp", the image data is first imported before it is sent in the long packet. In this case, FileOffset refers to the byte offset in the decoded image data and not the file. In addition, the decode format (e.g. RGB 8-8-8, 5-6-5, etc) can be selected via the ConfigWriteMemory dialog.

- **Block Partitioning –** A single WriteMemory command can be partitioned into a sequence of WriteMemory commands. The user configures this behavior via "Configure WriteMemory Commands…" in the Options menu.

Figure 9 shows the ConfigWriteMemory dialog which has the following controls:

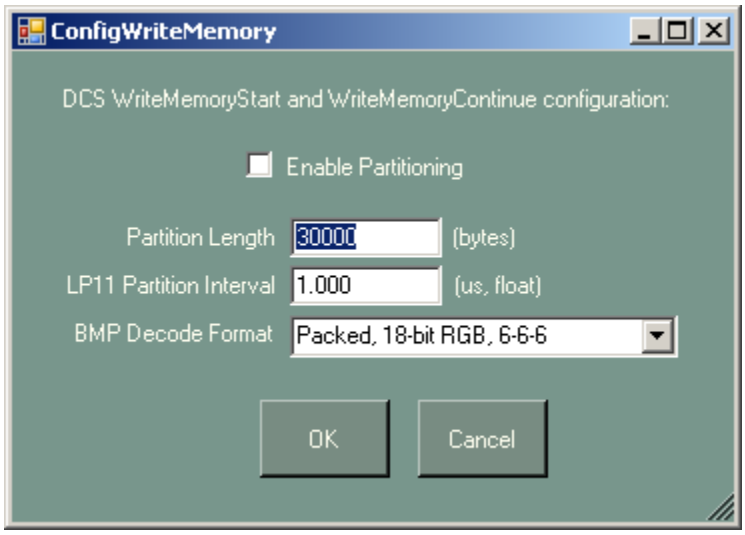

Figure 9 – Config WriteMemory Dialog

Enable Partitioning CheckBox – when checked, partitioning is enabled and WriteMemory commands will be deconstructed into a sequence of WriteMemory commands according to the parameters in the dialog. Otherwise, WriteLen values greater than 64 KB will return an error.

Partition Length – indicates the maximum length to use for component WriteMemory commands in the sequence.

LP11 Partition Interval – indicates the amount of time to spend in LP11 between each component WriteMemory in the sequence. Specify zero for no delay between component commands, which also means the command sequence will occur in a single HSDT or LPDT burst.

BMP Decode Format – specifies the format to decode files that have an extension of ".bmp". These formats include:

- Packed 16-bit RGB 5-6-5
- Packed 18-bit RGB 6-6-6
- Loosely packed 18-bit RGB 6-6-6
- Packed 24-bitRGB 8-8-8

As an example, consider a WriteMemoryStart command issued with the following configuration settings:

Partition Length = 30000
LP11 Partition Interval = 1.5 us

BMP Decode Format = Packed 24-bit RGB 8-8-8

If the file specified has extension ".bmp", FileOffset = 0, and WriteLen is 100000, PGRemote will send the following sequence:

WriteMemoryStart with RGB888 image data bytes 0-29999
LP11 for 1.5 us
WriteMemoryContinue with RGB888 image data bytes 30000-59999
LP11 for 1.5 us
WriteMemoryContinue with RGB888 image data bytes 60000-89999
LP11 for 1.5 us
WriteMemoryContinue with RGB888 image data bytes 90000-99999
LP11 for 1.5 us

3.10.4.5 File Command

The File Command is another way to define a custom command. In this case, the command references a file having one of two formats, an ASCII format or a binary format. The file defines the data packet data entirely, i.e. PGRemote does not compute or insert any bytes such as ECC, payload length or checksum. It is presumed the file data already has the correct packet format. Note that the file data could also consist of multiple DSI packets. When the file command is sent to the PG, PGRemote will send file data as a single HS or LP packet depending on DTMode.

The ASCII file format has the following syntax:

- A comment line begins with "//" and is ignored by the parser.
- A blank line that consists solely of spaces or tabs is ignored by the parser.
- The first non-comment, non-blank line should be "#ASCII".
- Subsequent lines define hex data bytes separated by white-space or commas.
- Any number of hex data bytes are allowed per line.

Example:

```
// This is an example ASCII file for the File Command
#ASCII
18 04 00 4f
87 4d 22 8c
```

If the parser cannot find the #ASCII token in the specified file, it is assumed to be a binary file. In this case, all bytes in the file are considered to comprise a data packet or series of packets to associate with the command name.

3.10.4.6 Conformance Testing (via File Command)

Additional in-line commands have been introduced into the ASCII file syntax for the File command to support low-level protocol conformance testing. These in-line commands, when present, modify default DPhy protocol behavior to introduce (generally) errors into the output stream.

In-line commands have the following syntax:

Format: #<cmd> <arg1> <arg2> …
Numeric arguments are always hexadecimal
One command per line
Commands must precede #ASCII command

The following in-line commands are supported:

#HS_SOT <lane0_SOT> <lane1_SOT> <lane2_SOT> <lane3_SOT>

Defines the SOT bit pattern for each lane
Only applies when File command is sent in HSDT mode
Bits are sent from SOT lsb first (i.e. default SOT is B8)
<lane_SOT> can range from 0-FF (all 8 bits are used), in hex format

#HS_END_BITS <value> <bitCnt>

Defines <bitCnt> bits from <value> to be added (as partial byte)
to end of HS packet
Only applies when File command is sent in HSDT mode
Bits are taken from <value> lsb first
<value> can range from 0-FF, in hex format
<bitCnt> can range from 1-7

#LP_LPDT <value>

Defines the LPDT command pattern
Only applies when File command is sent in LPDT mode
Bits are send from LPDT lsb first (i.e. default is 87)
<value> can range from 0-FF (all 8 bits are used), in hex format

#LP_RAW

File bytes define the entire LP sequence output
Ignores DTMode (data is always sent as LP)
Each byte pattern is output every TLPX time period
For each byte:
Lane 0 is defined by D[1..0]
Lane 1 is defined by D[3..2]
Lane 2 is defined by D[5..4]
Lane 3 is defined by D[7..6]
The two bits associated with a lane represent its LP state:
00 = LP00, 01 => LP01, 10 => LP10, 11 => LP11

#LP_RAW_WITH_CLOCK

File bytes define entire LP sequence output including clock LP signals.
Ignores DTMode (data is always sent as LP)
Each pair of bytes in pattern is output every TLPX time period
Within each pair of bytes:
Clk is defined byte D[1..0] of byte 1

Lane 0 is defined by D[1..0] of byte 2
Lane 1 is defined by D[3..2] of byte 2
Lane 2 is defined by D[5..4] of byte 2
Lane 3 is defined by D[7..6] of byte 2

The two bits associated with a lane represent its LP state:
00 = LP00, 01 => LP01, 10 => LP10, 11 => LP11

Examples:

#HS_SOT B7 B8 B8 B8

Causes a 11101101 bit sequence to be output as HS_SYNC for lane 0 and the correct 00011101 HS_SYNC to be output on lanes 1-3

#HS_END_BITS 18 5

Adds a 00011 5-bit sequence to the end of HS burst.data

#LP_LPDT B7

Sets the LPDT code to B7

#LP_RAW
#ASCII
FF EE DD CC DD EE FF

Sends the pattern: LP11, LP10, LP01, LP00, LP01, LP10, LP11 out on lanes 0 and 2 while outputting LP11 on lanes 1 and 3.

#LP_RAW_WITH_CLOCK
#ASCII
03 FF 03 EE 03 DD 03 CC 03 DD 03 EE 03 FF

Sends the pattern: LP11, LP10, LP01, LP00, LP01, LP10, LP11 out on lanes 0 and 2 while outputting LP11 on data lanes 1 and 3 and the clock lane.

Note also that the "Send Cmd To File" option allows any single command (non-macro, non DCS WriteMemory, non-video command) to be sent to a file rather than to hardware. The format of this file is ASCII text, suitable for editing and re-sending to hardware via the File command. For protocol testing, specifically, the user can edit any packet header fields, e.g. ECC, as well as the packet CRC bytes.

3.10.4.7 LPDelay and LPDelay (All Lanes)

The LPDelay and "LPDelay (All Lanes)" commands are low-level DPhy constructs that allows the user to assert a particular LP pattern on the bus for a specified period of time (or otherwise place the LP and HS transmitters in a high-impedance state for a specified period of time). As all commands are bracketed by LP11, this command is mostly useful in a macro sequence.

The LPDelay comand affects only the LP signal state of lane 0. Other lanes are held in LP11. Its arguments are as follows:

- **Delay (us)** – the length of time to hold the LP bus in the specified state. This value is floating point.
- **LP Driver En (0/1)** – set to 0 to set the drivers in high-impedance, 1 to enable the LP drivers.
- **LPValue[1:0] –** lane 0: set to 0 (LP00), 1 (LP01), 2 (LP10), or 3 (LP11)[16]

The "LPDelay (All Lanes)" command was recently added to allow the user to control the LP signal state on all four data lanes plus the clock lane:

- **Delay (us)** – the length of time to hold the LP bus in the specified state. This value is floating point.
- **LP Driver En (0/1)** – set to 0 to set the drivers in high-impedance, 1 to enable the LP drivers.
- **LPValue[1:0] –** lane 0: set to 0 (LP00), 1 (LP01), 2 (LP10), or 3 (LP11)
- **LPValue[3:2] –** lane 1: set to 0 (LP00), 1 (LP01), 2 (LP10), or 3 (LP11)
- **LPValue[5:4] –** lane 2: set to 0 (LP00), 1 (LP01), 2 (LP10), or 3 (LP11)
- **LPValue[7:6] –** lane 3: set to 0 (LP00), 1 (LP01), 2 (LP10), or 3 (LP11)
- **LPValue[9:8]** -- clk lane: set to 0 (LP00), 1 (LP01), 2 (LP10), or 3 (LP11)

3.10.4.8 Cmd Insertion Point

A new packet type has been introduced – "Cmd Insertion Point" – to allow command insertion during a looping macro. This feature is similar to command insertion during video mode. Its single parameter is a Delay value in microseconds, which represents a reserved period of time the bus will normally be in LP11. However, if a command is inserted, the command will be output during this time (and any remaining time after the command will still be spent in LP11).

The following describes command insertion behavior:

- To insert a command, the option "Enable Command Insertion" must be checked.
- A command sent while the macro is looping will attempt to be inserted.
- To send a subsequent command but not insert it, either stop the PG using the "PG Abort" button or uncheck the "Enable Command Insertion" option.
- If the inserted command requires more PG vectors than that reserved by the Cmd Insertion Point (via its Delay parameter), an error message will be reported.

[16] Note that until PGRemote version 1.0.46, the LPValue argument of the LPDelay command incorrectly output LP10 for (LPValue == 1) and LP01 for (LPValue == 2).

- When a command is inserted, PGRemote requests that the probe check to see if it can detect the insertion command code in the output stream. If it is not detected within ½ second, an error will be reported. To disable this check, select the "Disable command timeout" option. This option should be checked if the "Run PG on external event" option is enabled or the macro contains a "Wait Ext Event" command.

- If a command is inserted when the PG is waiting on an external event to continue output, the command is queued to be sent when output resumes and the insertion point is reached.

- If a macro contains multiple command insertion points, an inserted command will be output at the next encountered insertion point. Unless sequencing is controlled via external events, the user has no practical control over which command insertion point is used for the inserted command.

3.10.4.9 Read Commands and BTA

Read commands by default have their BTA (bus turn-around) argument set to "Yes". This argument can be set to "Yes" by the user for other commands as well. When BTA is set, the DPhy BTA signaling sequence is appended to the command, causing the slave device to obtain ownership of the bus subsequently. After the slave is done with the bus, it sends its own BTA sequence to return the bus back to the master.

When a command or macro having a BTA is sent, the P331 probe behaves exactly as desired. The probe pauses PG output, places its drivers in high-impedance until it receives a BTA response sequence, drives the BTA acknowledgement sequence, then resumes PG output driving data back onto the bus. In the event that no BTA response is received, the PG times out and drives LP11 back onto the bus.

3.10.4.10 BTA and BTA_WAIT

A BTA sequence can also be sent on its own, without accompanying data by selecting "Bus Turn Around" in the PktType field of the command. Alternatively, selecting "Wait For BTA" causes the P331 probe to put its outputs in high-impedance and wait for a BTA return sequence from the device under test.

Notes:

- These commands are most useful when embedded in a macro with other MIPI commands. Component commands in the macro following a BTA or BTA_WAIT are not output until the return BTA is recognized.
- In general, all non-video, non-looping commands will time out within about a second if a macro or command does not complete. To avoid timeout, the user can set the "Disable Command Timeout" option (Options menu). In this case, the user can press the PGAbort button to abort the command or macro if desired.
- While the PG and P331 are waiting for the return BTA response, the main window status bar will display "Waiting on BTA…".

- The RPC command equivalents to these two commands are RPCDefs.BTA and RPCDefs.WAIT_FOR_BTA.

3.10.4.11 Clock On and Clock Off

The "Clock On" and "Clock Off" commands are generally meant to be used within a macro (though they are not disallowed to be sent as individual commands). They give the user control over the clock lane, turning it on and off during component commands of the macro. Please section 3.7 for a more comprehensive discussion on clock lane behavior.

3.10.4.12 Escape Command

The Escape Command option lets the user send a DPhy escape command byte. The only argument is the command byte to send. Note that this command is different than a packet sent in LPDT mode. LPDT mode prepends the LP transmission with 0x87 to put the receiver in LPDT mode, and then sends the packet bytes.

3.10.5 Sending a Command to the PG

Once PGAppDotNet and command configuration has been set, commands may be sent to the PG (via the PGAppDotNet server). To send a command assigned to a button, merely left-click on the button. You can also select the command in the CmdName ComboBox and click on the Send button. Finally, a command definition does not have to be saved to be used. Just fill in the ComboBox and TextBox values as before but instead of saving, just click the Send button.

After the user requests a command to be sent, messages in the status bar will update progress. During this time, PGRemote performs the following tasks:

- The command is parsed and any necessary external file data imported (and decoded in the case of BMP files)
- MIPI protocol packets are built along with signaling structures (i.e. SOT, EOT, BTA, etc).
- Packet data is converted to PG vectors, and a PG program blocks and sequence is built and sent to the PGAppDotNet server.
- The PG is "Run" which downloads the PG program information to the PG3A module and initiates output.
- If the command is a looping video command, control is returned to the user at this point. Sending of the command is complete, though the PG continues to run.
- Otherwise, PGRemote waits for the PG to finish before returning control to the user. If the "Turn clock on/off each command" option is unchecked (available for the P331) the clock state remains in HS mode after the command is complete. Otherwise, the clock lane returns to LP11 state.

3.10.5.1 PG Abort

As described above, the sending of commands goes through several stages of processing, some of which may take significant time. Program download, in particular, can take tens of seconds to complete for video commands, when large

blocks of video data are sent to the PG3A. During this time, the user may click the PGAbort button (located in the bottom-right pane of the main window) to abort command sending. In addition, once a video command is running and the PG is actively looping video data, the PG Abort button may be clicked to stop the PG.

3.10.5.2 PG Restart

The PG Restart button in the bottom-right pane of the main window is used to resend the previous command, bypassing the potentially lengthy download process, useful, in particular, for video commands. When the PG is restarted, all timing configuration and parameters such as frequency, lane count, and clock start/stop mode from the original command are used (ignoring any changes since the command was sent).

3.10.6 PG Status

During operation, additional status is displayed in the lower-right pane of the PGRemote main window. This status is updated every second and includes the following:

- The PG run state (running, stopped)
- The clock lane state (running, stopped, or ULP).
- The data lane state (enabled, or ULP).
- Whether any LP contention has been detected since the last status reset
- Read response data received from the DUT after a BTA.[17]

Note that only the P331 can support the ULP lane state and contention detection and optionally maintain a continuous clock when the PG is stopped. In addition, only the P331 can acquire any read response data from the DUT.

3.10.6.1 Contention Detection

The P331 is equipped with contention and data receivers, whose thresholds can be set by the user in the PG Configuration dialog (see section 3.4.1 for more details). When the probe is outputting an LP high voltage and its receiver detects a voltage below the LP High contention threshold, an LP high fault is detected. When the probe is outputting an LP low voltage and its contention receiver detects a voltage above the LP Low contention threshold, an LP low fault is detected.

As these faults can occur on either LP signal wire this results in four possible contention faults displayed as status: "LP0H", "LP1H" (high faults) and "LP0L", "LP1L" (low faults). Contention flags, once detected, are held until a subsequent command is sent or the Status Reset button is pressed to clear state. Note also that the aggregate contention state is reflected on the Evt1 signal of the P331 back panel (see **Figure 2**), allowing for instrument trigger on the onset contention.

[17] Note: LPDT response acquisition requires P331 firmware version 1.6 or above and assumes DUT response data is sent via an LPDT command.

3.10.6.2 DUT Response

When a command or macro containing a BTA is sent, the P331 (with firmware version 1.6 or above) monitors the link for an LPDT response command from the DUT. The DUT response (if any) is displayed in the status pane of the main window as the string "DUT Resp: " followed by hex data bytes (up to 12). Note that the first byte is special, containing only the first two bits captured from the DUT, and should always be 01h. These bits represent the escape entry bit pattern (LP10, LP00, LP01, LP00). Subsequent bytes represent bits obtained after escape entry, starting with the escape command, which is expected to be 87h for an LPDT command. Thus, the first header byte of the DUT response command will be in the third byte of the displayed DUT response data.

The DUT response buffer is reset every time a PGRemote command is sent, but this could consist of a macro with multiple read commands. In the P331 and P332, the response buffer is very small (12 bytes) and thus is not suited for capturing multiple responses from the DUT during a macro. On the other hand, in the P338, the response buffer is relatively large (4 Kbytes) and thus can capture a significant amount of data from multiple reads.

In the P338, to facilitate parsing of data from the DUT response buffer, an additional flag is added (0x100) to the data bytes to indicate a new BTA. Thus, the first response byte from each command should now start with 0x101 (rather than 0x01 for the P331/P332).

Note, there are two RPC commands that can be used to obtain the DUT response:

- GET_DUT_RESPONSE can be used programmatically to obtain the last response from the DUT.

- SAVE_DUT_RESPONSE can be used either in a script file or programmatically to save or append the last response from the DUT to a text file.

Please see Appendix B for command syntax.

3.10.7 Saving/Restoring Command Configurations

A default.cfg file is provided with PGRemote installation that has several single parameter commands defined (they all assume the Virtual Channel of the slave device is 0). Once you add new command definitions, you can save your command configuration, command set, and button assignments in a configuration file. Use the "File->Save XXX.cfg As..." menu option and specify the name of the file to save to. Similarly, you can load a configuration file with the "File->Load..." menu option. Also, the last configuration file saved before exit is restored on relaunch.

3.11 Macros

A macro is a sequence of non-video commands, grouped and named as a single, new command. Once defined, they can be assigned to command buttons and sent to the PG like any other command. When viewing and creating macros, an additional pane in the

main window is shown that displays the component commands of the macro and allows basic macro editing (see **Figure 10** below).

3.11.1 Defining a new macro

To define a new macro, click on the "Start Macro" button at the bottom right of the main window. This button will change its caption to "End Macro" and will cause the macro pane to appear with no component commands in the macro commands list-box (unless the current command is a macro command, in which case the component commands of the macro will be shown – see the next section). In this mode (called macro mode), commands that are sent either via a command button, via the Send button at the top of the main window, or via an RPC programmatic command are entered into the macro instead of sent to the PG. **Figure 10** shows an example of a macro in edit mode.

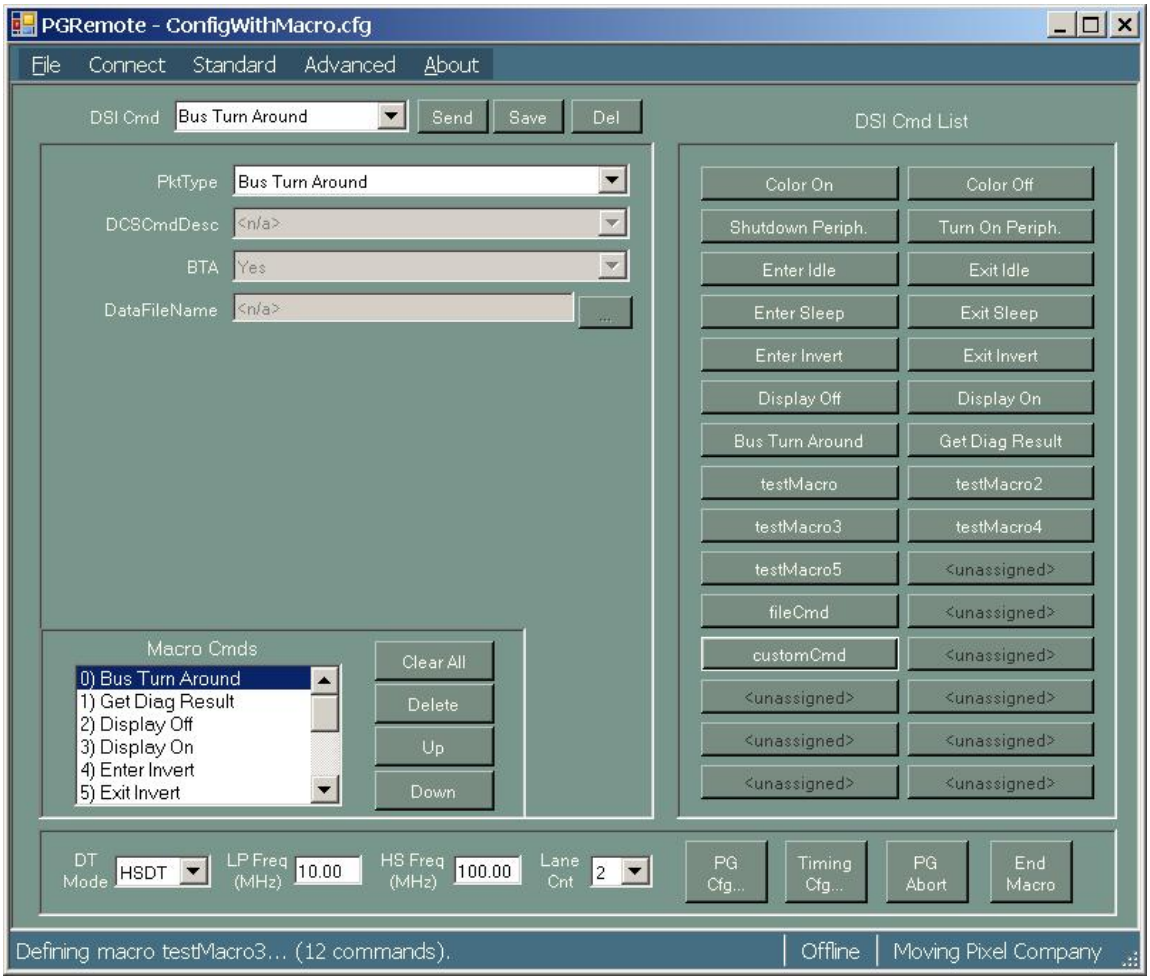

Figure 10 – Macro Editing

Commands are inserted into the macro at the location highlighted in the macro pane. When inserted, the "DSI Cmd" field name is used to label the compenent command of the macro. The user can scroll and select a command (or the blank line at the end of the

command list) to mark the next command insertion location. Selecting a command also displays its arguments for viewing and modification.

Four buttons are available in the macro pane to manipulate the component commands of a macro. **Clear All** removes all the commands from the macro. **Delete** removes the selected command from the macro. **Up** moves the selected command up within the command sequence and **Down** moves the selected command down within the command sequence.

When macro definition is complete, the user clicks on the "End Macro" button, bringing up a small dialog to name the macro, save it, and exit macro mode (the user may also **Cancel** to return to macro editing or **Discard** to discard the macro edits entirely.

As with command definition, if the user specifies the same name when saving a macro, the user will be asked for permission to replace the existing macro definition.

3.11.2 Editing an Existing Macro

To edit and existing macro, the user has two options. If the macro is assigned to a command button, the user may right-click on it and select "Edit Current Command…". Alternatively, the user can select it in the command combo-box at the top of the main window then click on the "Start Macro" button. The macro can then be edited as normal and saved by clicking on the "End Macro" button. The name field will be filled in with the original macro name and the user can click OK and confirm overwriting the existing macro.

3.11.3 Copying a Macro

To copy a macro, simply load a macro for editing, click on the "End Macro" button, and replace the original name with a new name.

3.11.4 Editing a Component Command

To edit a component command of a macro, perform the following:

1) Open a macro for editing.
2) Click on a component command in the macro to retrieve its argument settings.
3) Change the argument settings as desired.
4) Click the 'Send' button to insert the modified command into the macro (both the new command and old command will now be in the macro list)
5) Click the 'Delete' button to delete the old component command.
6) Click the 'End Macro' button and Click 'OK' to save the modified macro.

3.11.5 Allowed Component Commands

Any single LP or HS command can be inserted into a macro. In addition, here are notes on inserting special command types in to a macro:

- Component commands with BTA enabled are allowed in macros and behave as expected. The PG twaits for a return BTA before continuing with the remaining component commands in the macro.
- WriteMemoryStart and WriteMemoryContinue commands are fully supported in macros, including BMP file decoding and command partitioning to support large WriteLen counts.
- Macros cannot contain other macros; that is, macro nesting is not allowed. However, macros can nonetheless be added to other macros, in which case the component commands of the macro being added are themselves duplicated and added to the macro being edited.
- Unlike all other commands, video mode commands are treated differently when added to a macro than when sent directly. In a macro, video mode commands simply add a single video data packet to the macro (unlike the entire video frame that is built when sent directly). Other differences from video mode usage:

  1) The entire contents of the given file are put in the video packet
  2) The FrameCnt parameter is ignored
  3) The file argument is assumed to be in binary format (i.e. BMP file decoding is not supported).

  Thus, to manually build an entire video frame into a macro would require a binary file for each active line. This capability is more suited for programmatic macro building using RPC commands.

3.11.6 HS Component Command Behavior

When macros are parsed by PGRemote, the application groups HS component commands (that don't have BTA enabled) into a single HS burst. To send HS component commands in its own burst, the user must put an LPDelay call between each HS command.

3.11.7 DPhy Clock Lane Behavior

In macros, clock lane behavior can be modified from its default behavior using the Clock On and Clock Off commands. These commands can be inserted at any position in a macro to turn the clock land on and off (according to the procedure defined in the DPhy standard). Please see section 3.7.1 for more details.

3.11.8 Additional Notes on Macro Behavior

Below lists some final miscellaneous notes on PGRemote behavior when working with macros:

- Macro names must be distinct from command names. PGRemote will not allow a macro to be saved with the same name as a command and vice versa.
- Note that once commands are added to a macro, they are duplicated and no longer are associated with the original command. Specifically, if the original command is then modified and saved, the duplicate command in the macro having the same name does NOT change. Moreover, although it would be a confusing practice to

do this, multiple component commands may be added to a macro that have the same name but different arguments.

- Also, sending an RPC script that contains SEND_MIPI_CMD calls while PGRemote is in macro mode adds the commands to the macro. Note that other RPC command types in the script are executed (if possible in macro mode) at the time the script is sent. These command types are NOT saved in the macro.

4 PGRemote RPC (Remote Procedure Calls)

To facilitate automated testing, PGRemote supports incoming RPC requests from other applications via a Microsoft .NET TCP server port. This section describes details of the communication interface, supported commands, and example client code provided with installation of the PGRemote application. It is assumed that the user is conversant with programming in the .NET environment or otherwise knowledgeable about interfacing between his preferred programming environment (e.g. LabView) and .NET.

Before skipping this chapter entirely, please note that RPC commands can be embedded in a text script file and run from the PGRemote GUI. In this case, no external programming is required. You will find script files are a very powerful tool to configure and control PGRemote. Please see section 4.4 for more details.

4.1 Using PGRemote as an RPC Server

To enable PGRemote as an RPC server, simply select the "Enable RPC" menu option in the Connect menu. If this option is not already checked, a simple dialog will appear requesting the port number to use for incoming RPC requests. You will need to use a port number that is not already in use by your system and other applications. This number should be between 1024 and 65525, though the top end of that range (greater than 49152) is best as there are no registered ports. For more information, go to the following link:

http://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers

To disable PGRemote as an RPC server, uncheck the "Enable RPC" menu option.

When PGRemote is enabled as an RPC server, the user can interact with the GUI as normal. In addition, RPC calls from other applications can interact with the application as well, able to perform many of the functions programmatically that the user can perform using the GUI.

Notes:

- Currently there is no access control to prevent multiple applications from connecting and making RPC calls simultaneously to the PGRemote server even as the user is using the application, which, of course, may cause confusing behavior.
- Where a user initiated action would normally elicit a message box on PGRemote, usually a warning or a confirmation, the equivalent RPC call will not do so. Instead, the operation proceeds as if the user had responded affirmative to the message.
- RPC calls block until PGRemote has completed processing them. In some cases, this can take many seconds, especially when reconfiguring the PG and probes or sending a MIPI command, especially video frames. For client applications that require a responsive interface, RPC calls should be made from a separate thread.

4.2 PGRemoteRPCClient Library

To facilitate the user in building RPC client programs to control PGRemote, a DLL is provided (PGRemoteRPCClient.dll). This library is written in C# and provides classes to encapsulate error codes, command definitions, command field definitions, and a few simple calls to establish a connection to the PGRemote server and send commands. These classes are contained in the namespace "PGRemoteRPC" and are described below.

Note: for those interfacing to the PGRemoteRPCClient DLL from a language other than C# or an environment different than .NET and you have suggestions how to make the DLL more usable for your needs, please contact the Moving Pixel Company. This may be necessary especially with respect to function signatures, which currently take advantage of C# generic types and may not be easily portable to other languages or environments.

4.2.1 PGRemoteRPCClient Class Overview

The PGRemoteRPCClient library contains four classes:

- PGRemoteClient – main client instance to connect and send commands to PGRemote server
- RPCErrs – error code definitions
- RPCCmds – RPC command code definitions
- RPCDefs – RPC command parameter definitions

These classes are further described in the following sections.

4.2.1.1 PGRemoteClient Class

The PGRemoteClient Class represents the main class of the library, providing an interface to connect to the PGRemote server, send RPC commands, and obtain results.

The client constructor takes not arguments and is simply created with the call:

```
PGRemoteClient client = new PGRemoteClient();
```

Next, should be a call to connect to a PGRemote server. The server should be running and enabled for RPC as described in section 4.1. As an example, the following call connects to a PGRemote server located on the local host machine and enabled for RPC at port 2799:

```
int rc = client.Connect("", 2799);
```

When finished sending RPC commands, disconnect from the server with

```
int rc = client.Disconnect();
```

4.2.1.2 RPCErrs Class

The RPCErrs class contains definitions for the possible error codes that can occur in the PGRemote server (see Appendix A). However, note that only a subset of the errors in

this class will be returned by an RPC call. No comprehensive documentation of these errors is provided currently – the names are meant to provide a more descriptive indication of the error that occurred. In many cases, the error code returned is also supplemented with an error message that is returned as well by the RPC call.

A few error codes deserve further comment:

FAIL: used as a generic catch-all failure code.

LOCAL_FAILURE: indicates that an error occurred on the client side (before reaching the PGRemote server). Often this error is associated with an error marshaling unexpected or incorrect command arguments.

CONTROL_IS_DISABLED: returned when setting the underlying value of a control that is disabled in PGRemote, which indicates the feature is not currently available or supported in the current configuration.

NEED_START_EDIT_COMMAND: indicates a configuration command was sent without a prior START_EDIT_CONFIG command.

4.2.1.3 RPCCmds Class

The RPCCmds class contains definitions for all the possible RPC commands that can be sent to the PGRemote server (see Appendix B). RPC commands can be categorized as follows:

MIPI Command: commands that send a MIPI command.

PG and Probe Configuration: commands that set any of the PG Configuration dialog settings (e.g. voltages, delays, etc.), as well the LP/HS frequency, Lane Count, and DT Mode settings.

Menu: relevant commands from the Connect, Standard, and Options menus.

Timing Configuration: commands that set any of the Timing Configuration dialog settings (e.g. frame component dimensions, video timing and playback options).

PG Operational: commands to abort and restart the PG.

PG Status: commands to retrieve status such as PG and probe connectivity, as well as PG, lane clock, and contention state.

Support: miscellaneous support commands to compute ECC and CRC values on user data.

Notably, the current RPC interface does not provide any commands relating to defining named MIPI commands, saving them, and assigning them to buttons.

To send MIPI commands, use the MIPICmd RPC call. All other command categories use one of three signature variants of the PGRemoteCmd call. See section 4.2.1.4 for more details.

4.2.1.4 RPCDefs Class

The RPCDefs class overview contains constants to be used for RPC command parameters. Predominantly, this class contains all the MIPI command codes that can be used to send MIPI commands. More specifically, this class includes the following types:

- DSI Command Codes
- CSI Command Codes
- DCS Command Codes
- DPhy Command Codes
- Data Transfer Modes
- Probe Types
- MIPI Standards
- Menu Options
- Contention Mask Bits

See Appendix C for details.

4.2.2 Sending a MIPI Command

Because the argument structure used for sending MIPI commands is different than for other RPC commands, a separate RPC call (MIPICmd) is used for this command class. The MIPICmd RPC call has the following signature:

```
public int MIPICmd(int cmdCode, int DCSCmdCode, bool BTA, int
DTMode, int VC, int arg1, int arg2, int arg3, string fn, byte[]
data, ref string errMsg, ref string statusMsg);
```

Its parameters are a superset of what the user sees in the PGRemote GUI, where the cmdCode dictates which parameters are used and which are unused. For example, none of the short packet form DSI commands use the "fn" parameter and only one command (SET_SCROLL_AREA) uses the "arg3" parameter. **Table 4** describes the parameters of the MIPICmd library call.

Table 4 – MIPICmd Parameters

cmdCode	The RPC command code
DCSCmdCode	The DCS command code (applies only to DCS commands)
BTA	Requests a BTA sequence following the MIPI command
DTMode	Specifies the data-transfer mode for the command
VC	The MIPI virtual channel for the command (0-3)
arg1	cmdCode specific argument
arg2	cmdCode specific argument
arg3	cmdCode specific argument

fn	Specifies the file name, path relative to the PGRemote application, to use for the command (applies only to commands that use file argument). If the data argument is non-null, this field is ignored.
data	Specifies a data buffer to use for the command instead of a file (applies only to commands that use file argument). If non-null, the fn argument is ignored.
errMsg	If return code is negative, contains the error message string
statusMsg	Returns the last text displayed in the PGRemote status bar after the command is sent
return code	The return code from the command. A negative return code indicates an error from the RPCErrs class. A zero return code indicates success for most commands. Commands that return a value, e.g. GET_PG_PROBE_TYPE return a non-negative value for their result.

To give some examples, the following command causes PGRemote to send a Generic Short Write DSI command with 1 parameter (0xaa) as a high-speed packet on the DSI link:

```
rc = client.MIPICmd(RPCDefs.GENERIC_SHORT_WRITE, 0, false,
    RPCDefs.DT_HS, 0, 1, 0xaa, 0, "", null,
    ref errMsg, ref statusMsg);
```

In this case, several of the MIPICmd parameters are unused by PGRemote, specifically DCSCmdCode, BTA, arg3, fn, and data.

RPC calls mimic the functionality of PGRemote when used through the GUI. For example, to send a video frame using a BMP file as the source and the current timing parameters of PGRemote, you could use the following call:

```
rc = client.MIPICmd(RPCDefs.PACKED_PIXEL_STREAM_565,
    0, false, RPCDefs.DT_HS, 0, 0, 0, 0,
    "c:\\pictures\\test12x12.bmp", null,
    ref errMsg, ref statusMsg);
```

If you don't want PGRemote to parse the file as a BMP image, provide a filename without the ".bmp" extension. The bytes will then be interpreted as raw video data.

An alternative available to a program using an RPC call instead of a user using the GUI, is to make use of the "data" parameter of MIPICmd:

```
rc = client.MIPICmd(RPCDefs.PACKED_PIXEL_STREAM_565,
    0, false, RPCDefs.DT_HS, 0, 0, 0, 0,
    "", imageData, ref errMsg, ref statusMsg);
```

In this case, the imageData buffer containing data in the appropriate RGB565 format is used instead of a file. This allows easy programmatic construction of video frames for testing.

Another example causes PGRemote to build and send a video frame as a sequence of DCS WriteMemoryStart and WriteMemoryContinue commands.

```
rc = client.MIPICmd(RPCDefs.DCS_LONG_WRITE,
    RPCDefs.WRITE_MEMORY_START, false, RPCDefs.DT_HS,
    0, 0, 921600, 0,
    "c:\\pictures\\rgbstripes640x480.bmp",
    null, ref errMsg, ref statusMsg);
```

This call requests that PGRemote read in the given BMP file (as RGB888 format), break 921,600 bytes of it (640 x 480 x 3) into ~64 KB segments and send successive WriteMemoryStart / WriteMemoryContinue commands in a single PG sequence.

Note the RPC script command equivalent to this call is SEND_MIPI_CMD.

4.2.3 Sending a MIPI Command With Errors

An extended version of MIPICmd called ImpairedMIPICmd is provided that allows the user to set the ECC and/or CRC fields of the command in order to cause receive errors in the device under test. In addition, this command allows the user to provide a byte offset within the constructed packet and an XOR byte mask to be used to introduce errors in the packet at the given byte offset.

Below is the calling syntax of the ImpairedMIPICmd call:

```
public int ImpairedMIPICmd(int cmdCode, int DCSCmdCode,
        bool BTA, int DTMode, int VC, int arg1, int arg2, int arg3,
        int customECC, int customCRC, int errByteOff,
        int errByteMask, string fn, byte[] data,
        ref string errMsg, ref string statusMsg)
```

The command is similar to the MIPICmd call (see previous section for details), except for the addition of four parameters. These additional parameters are described below:

Table 5 – Additional ImpairedMIPICmd Parameters

customECC	The 8-bit ECC value to use for the packet. Set to -1 to use the correct computed ECC.
customCRC	The 16-bit CRC value to use for long packets. Set to -1 to use the correct computed CRC.
errByteOff	Specifies the byte offset, starting from the first byte in the packet header, to the byte to XOR with the errByteMask to introduce an error in the packet. Set to -1 to disable this function.
errByteMask	Specifies the XOR error byte mask to use to use at errByteOff in the packet.

The errByteOff, as described above, starts from the beginning of the packet. Thus, a value of 0 references the DataID, a value of 3 references the ECC, a value of 100 references byte 96 of the packet payload. If a value is given greater than the packet length, it is ignored.

Note that the customECC and customCRC are set in the packet AFTER the errByteMask is applied. So, for example, if errByteOff was set to 3 and customECC was set to 45, the ECC in the packet would be 45. However, if the customECC was set to -1, the computed ECC for the packet would be XORed with errByteMask to determine the ECC value put in the packet header.

Note the RPC script command equivalent to this command is SEND_IMPAIRED_MIPI_CMD.

4.2.4 Sending Other RPC Commands

Aside from sending a MIPI command, most other RPC commands make the use of generic RPC call PGRemoteCmd, which has three function signatures, accommodating 0, 1, and 2 command arguments respectively:

```
public int PGRemoteCmd(int cmdCode, ref string errMsg,
                       ref string statusMsg)

public int PGRemoteCmd<T1>(int cmdCode, T1 arg1,
                       ref string errMsg, ref string statusMsg)

public int PGRemoteCmd<T1, T2>(int cmdCode, T1 arg1, T2 arg2,
                       ref string errMsg, ref string statusMsg)
```

In addition, one other call may be used, specifically for calls that return more than a single integer return code (currently only GET_DUT_RESPONSE requires this call)

```
private int PGRemoteQuery(int cmdCode, ref byte[] respVal,
                       ref string errMsg, ref string statusMsg)
```

Which call should be used depends on the RPC command (see Appendix B for RPC command descriptions). For example, to set the HS frequency to 300 MHz, the single-argument signature call is used, e.g.:

```
rc = client.PGRemoteCmd(RPCCmds.SET_HS_FREQ,
                        (float)300e+6,
                        ref errMsg, ref statusMsg);
```

But to set an option, the dual-argument signature call is used, e.g.:

```
rc = client.PGRemoteCmd(RPCCmds.SET_OPTION,
                        RPCDefs.OPT_LOOP_NON_VIDEO_COMMANDS, true,
                        ref errMsg, ref statusMsg);
```

Note that because the PGRemoteCmd calls have generic arguments, any type may be passed in successfully. However, a side-effect of this is that no implicit type casting is performed (e.g. double-to-float, float-to-int, int-to-byte, etc.). Consequently, because PGRemote expects command arguments to be a specific type, it will return an ARGTYPE_MISMATCH error if the wrong type parameter is supplied.

4.2.4.1 Alternate Command Interface For Labview

Some languages or programming environments cannot support the generic style argument passing (which is a Microsoft C# construct). Accordingly, the PGRemoteRPCClient DLL supports the following alternate procedure calls:

```
public int PGRemoteCmdF(int cmdCode, float arg1,
                  ref string errMsg, ref string statusMsg);
public int PGRemoteCmdI(int cmdCode, int arg1,
                  ref string errMsg, ref string statusMsg);
public int PGRemoteCmdIF(int cmdCode, int arg1, float arg2,
                  ref string errMsg, ref string statusMsg);
public int PGRemoteCmdII(int cmdCode, int arg1, int arg2,
                  ref string errMsg, ref string statusMsg);
public int PGRemoteCmdS(int cmdCode, string arg1,
                  ref string errMsg, ref string statusMsg);
```

These calls have explicit argument types and can be used instead of their generic equivalents. The naming convention is such that 'F' stands for a floating point argument, 'I' stands for and integer argument, and 'S' stands for a string argument.

4.2.5 RPC Notes

The behavior of most commands, their usage, and arguments are self-explanatory, but certain modes and commands some deserve additional comment.

4.2.5.1 Setting MIPI Standard

Before sending any MIPI commands, make sure to send the SET_MIPI_STANDARD command to select which command set is available (currently CSI or DSI). Otherwise, the CMD_STANDARD_MISMATCH error will be returned for calls not supported by the current standard.

4.2.5.2 Inserting Commands During Video Playback

Command insertion while video is playing is achieved via RPC similarly to when using the GUI. Simply, set the enable-command-insertion option, send a continuous video command, and then send other (short) non-video MIPI commands. They will be sent during the vertical blanking of one of the looping video frames. Note that if this option is not set, subsequent commands to a looping video command will stop the PG first. This happens automatically since, as noted earlier, all message box queries from PGRemote are assumed to be answered affirmatively.

4.2.5.3 Sending Configuration Commands

Because PG and probe configuration requires several seconds to complete, RPC configuration commands must be bracketed with START_EDIT_CONFIG and

END_EDIT_CONFIG. Configuration commands only take effect after END_EDIT_CONFIG has been sent, allowing configuration to be performed only once after many commands have been sent. Multiple start/end sequences are allowed and there is no restriction on which subset of configuration commands can be present in each sequence.

For example, the following sequence would be used to set the DTMode, LP and HS frequencies, and lane count:

```
// start setting configuration parameters
rc = client.PGRemoteCmd(RPCCmds.START_EDIT_CONFIG,
                  ref errMsg, ref statusMsg);

rc = client.PGRemoteCmd(RPCCmds.SET_DT_MODE, RPCDefs.DT_HS,
                  ref errMsg, ref statusMsg);
rc = client.PGRemoteCmd(RPCCmds.SET_LP_FREQ, (float)7.6e+6,
                  ref errMsg, ref statusMsg);
rc = client.PGRemoteCmd(RPCCmds.SET_HS_FREQ, (float)300e+6,
                  ref errMsg, ref statusMsg);
rc = client.PGRemoteCmd(RPCCmds.SET_LANE_CNT, 4,
                  ref errMsg, ref statusMsg);

// done setting configuration parameters... send to PG and probe
rc = client.PGRemoteCmd(RPCCmds.END_EDIT_CONFIG,
                  ref errMsg, ref statusMsg);
```

4.2.5.4 Defining and Sending Macros

Similarly, macros are defined by bracketing their component commands with START_MACRO and SEND_MACRO. The START_MACRO call puts PGRemote in macro-mode and initializes to build a new macro. All MIPI commands that are subsequently sent are added to the current macro (rather than forwarded to the PG for playback). Then, when the SEND_MACRO call is made, macro-mode is exited, and the macro is sent to the PG for playback. Thus, this mode is useful for packing multiple commands into a single sequence for playback without large time gaps in between them.

Unlike defining and editing macros using the GUI, macros defined via RPC have no name and are not stored or saved by PGRemote. Each time a macro is sent, it must be first defined. The last macro defined, however, may be repeatedly sent using SEND_MACRO.

Only MIPI commands may be components of a macro (i.e. commands sent via MIPICmd or ImpairedMIPICmd and not PGRemoteCmd). In addition, as with defining macros using the GUI, LP and HS packet types may be mixed. Note that, while in the PGRemote GUI sending packed pixel stream commands result in video mode being invoked (and full frames built as a result), when these commands are sent in a macro, they result in a single packed pixel stream packet being sent. Thus, conceptually, RPC macros can be used to build and send full video frames.

The following sequence defines and plays a simple macro:

```
// begin macro definition
rc = client.PGRemoteCmd(RPCCmds.START_MACRO,
                  ref errMsg, ref statusMsg);

rc = client.MIPICmd(RPCDefs.COLOR_MODE_ON, 0, false,
                  RPCDefs.DT_HS, 0, 0, 0, 0, "", null,
                  ref errMsg, ref statusMsg);
rc = client.MIPICmd(RPCDefs.TURN_ON_PERIPHERAL, 0, false,
                  RPCDefs.DT_HS, 0, 0, 0, 0, "", null,
                  ref errMsg, ref statusMsg);
rc = client.MIPICmd(RPCDefs.DCS_SHORT_WRITE,
                  RPCDefs.ENTER_NORMAL_MODE, false,
                  RPCDefs.DT_HS, 0, 0, 0, 0, "", null,
                  ref errMsg, ref statusMsg);

// end macro definition and send to PG
rc = client.PGRemoteCmd(RPCCmds.END_MACRO,
                  ref errMsg, ref statusMsg);
```

4.2.5.5 Using the Custom and File Commands

Three generic calls are available for the user to define arbitrary commands.

First are the CUSTOM_COMMAND and CUSTOM_LONG_COMMAND calls, which can be used to build arbitrary packets with proper DSI/CSI packet structure. Both have arguments consisting of the DataID to use and a data array containing the payload bytes for the command. For the CUSTOM_COMMAND call, if the payload array is less than or equal to two bytes, a 4-byte short packet is built with the correct ECC. Otherwise, a long packet is built with the correct length field, ECC, and checksum given the DataID and payload data. For the CUSTOM_LONG_COMMAND call, a long packet format is always used, even for zero, one and two byte data array lengths.

For example, the following code sends a 2-byte short packet with DataID = 0xfd:

```
// send custom short packet
byte[] data = new byte[2];
data[0] = 0x1b;
data[1] = 0x2c;
rc = client.MIPICmd(RPCDefs.CUSTOM_COMMAND, 0, false,
                  RPCDefs.DT_HS, 0, 0xfd, 0, 0, "", data,
                  ref errMsg, ref statusMsg);
```

And this code sends a 100-byte long packet with DataID = 0xfd:

```
// send custom long packet
byte[] data = new byte[100];
for (int inx = 0; inx < 100; inx++)
      data[inx++] = (byte)inx;
rc = client.MIPICmd(RPCDefs.CUSTOM_COMMAND, 0, false,
                  RPCDefs.DT_HS, 0, 0xfd, 0, 0, "", data,
                  ref errMsg, ref statusMsg);
```

The third generic call is FILE_COMMAND. When sent through the GUI, this call reads in a file that contains the entire packet data to send, including header and checksum fields, if even present. The RPC command can also reference a file, in which case the command behaves identically as if sent using the GUI. Alternatively, the RPC call can provide a data array instead of a file name, allowing for arbitrary packet data to be sent programmatically without the use of data files. The following code is an example:

```
// send data as FILE_COMMAND (short packet)
// The DataID = 0xfd, with arguments 0xaa and 0xbb,
// and an erroneous ECC = 0xcc
data = new byte[4];
data[0] = 0xfd;
data[1] = 0xaa;
data[2] = 0xbb;
data[3] = 0xcc;
PE(client.MIPICmd(RPCDefs.FILE_COMMAND, 0, false,
                  RPCDefs.DT_HS, 0, 0, 0, 0, "", data,
                  ref errMsg, ref statusMsg));
```

4.2.5.6 Using the USER_WAIT Command

Often an RPC script is used for test automation. To facilitate using a single script to perform a sequence of tests, the USER_WAIT command is provided. When this command is sent, a message box is displayed with the supplied string message. For example, the message might read "Click OK to perform test #6".

While the application is waiting, the user can perform whatever actions are necessary to retrieve the results from the previous test and set hardware up for the subsequent test. When ready, the user can click OK to continue executing script commands or CANCEL to abort script processing.

4.3 TestRPC Project

When PGRemote is installed, a simple Visual Studio 2008 project written in C# is included that demonstrates use of the RPC interface. The project is called TestRPC and is located in the PGRemote application directory:

c:\Program Files\TMPC\PGRemote

The Main procedure in the project is in Program.cs and simply connects to the PGRemote server which is assumed to be running on the local host and enabled for RPC at port 2799. In addition, PGAppDotNet should be running, either in Offline mode or connected to a PG module with a P331 probe. Once connected, the routine simply progresses through almost all of the RPC commands available. This code doesn't perform anything particularly useful and is intended to be run in the debugger, stepping or running with breakpoints set at locations of interest.

The only other point of interest in the TestRPC project is to note in the References section the reference to the PGRemoteRPCClient DLL, which is located in the bin/Debug directory (along with sample bmp and data files used by some of the RPC calls).

4.4 RPC Script Files

RPC script files are text files with embedded RPC commands. They are very useful since the RPC script command can be built and saved as a named command and assigned to buttons like any other command. As RPC commands can assign almost all settings in PGRemote and can send almost all commands to the PG, it allows for powerful reconfiguration and testing capabilities at the push of a single button.

This section describes the syntax used in RPC script files. Please note there is now an easy method for automated script file generation. For more information, please read section 3.7.6 on script recording.

Please note that to use RPC scripts you do not need RPC enabled on PGRemote and you do not need a programming environment or DLL libraries. All you need is a text editor.

An RPC script file has the following syntax:

- A comment line begins with "//" and is ignored by the parser.
- A blank line that consists solely of spaces or tabs is ignored by the parser.
- A command line begins with the '#' character, followed by a space, followed by an RPC command plus its arguments. Only one RPC command may be defined per line.
- String constants that match any of the definitions in Appendix B or Appendix C may be used instead of numeric values for any command or its arguments.
- String constants are case insensitive (e.g. send_mipi_cmd is equivalent to Send_Mipi_Cmd is equivalent to SEND_MIPI_CMD)
- Numeric values are assumed to be decimal, unless appended with an 'h', in which case they are parsed as hex. So, "10" is interpreted as 10 decimal, "10h" is interpreted as 16, "3f" results in a parsing error, and "3fh" is interpreted as 63.
- File names should have double-quotes, e.g. "c:\rpcScript.txt". A null file should be indicated with a pair of double-quotes, e.g. "".
- or use the string "null" (case-insensitive) to specify no file name.
- Commands that specify a data buffer should use the string "null" (case-insensitive) to indicate a null buffer or simply list data values at the end of the command to indicate buffer values. Currently, RPC commands cannot span more than one line in the RPC script file so all data buffer values must be included on the command line.

RPC script files need to adhere to the same rules as RPC programming, and specifically certain command types have some additional requirements. Specifically, configuration commands must be bracketed by START_EDIT_CONFIG and END_EDIT_CONFIG calls, macros must be bracketed by START_MACRO and SEND_MACRO calls. Please see sections 4.2.5.3 and 0 for more information. An example RPC script file is provided for you in the application directory, by default:

c:\Program Files\TMPC\PGRemote\TestRPCScript.txt.

As described in previous sections, an RPC call is either a MIPI command or a PGRemote command and each command type has its own library call when using RPC (MIPICmd or PGRemoteCmd). Similarly, in an RPC script file the different call types have a different argument structure and mimic the arguments for the programmatic RPC calls.

Thus, if using the SEND_MIPI_CMD, you are required to provide the 10 arguments for the MIPICmd call as described in **Table 4**. For example, to send a HS generic short write command with two arguments with values 0x10 and 0x20, you could use the following script line:

```
# SEND_MIPI_CMD GENERIC_SHORT_WRITE 0 0 DT_HS 0 2 10h 20h "" NULL
```

In this case, you must refer to Appendix C to obtain the command argument definitions for arg1, arg2, and arg3 of the GENERIC_SHORT_WRITE command. Note the string literals "SEND_MIPI_CMD", "GENERIC_SHORT_WRITE", and "DT_HS" are not required and could be replaced by their numeric equivalents if desired.

If you wanted to cause a receive error in your DUT by changing the ECC to 45 (for example) you might use the following:

```
# SEND_IMPAIRED_MIPI_CMD GENERIC_SHORT_WRITE 0 0 DT_HS 0 2 10h 20h
45 -1 -1 0 "" NULL
```

A PGRemote command has a variable number of arguments and argument types depending on the specific command. Thus, for example, you can set the MIPI standard to CSI or the video timing line time with the following commands:

```
# SET_MIPI_STANDARD STD_CSI
# SET_TIMING_LINE_TIME 5e-6
```

Just remember if you are setting any of the configuration commands you must bracket the calls with start and end configuration commands, e.g.

```
# START_EDIT_CONFIG
# SET_HS_FREQ 120e+6
# SET_LP_FREQ 15e+6
# SET_LANE_CNT 4
# END_EDIT_CONFIG
```

Finally, once a script file is defined you can execute it by simply selecting RPC Script from the PktType list-box and type in or browse for the script file name. Then, like any other command, you can click the Send button to execute the script file, or type in a name to save it as a named command and then assign it to a button.

5 Using the P338 Probe

5.1 Overview

The P338 probe is the newest MIPI probe made by the Moving Pixel Company. It has 8 data lanes and two clock lane outputs, grouped as two 4-lane DPhy links. When both links are active in dual-interface modes, the P338 can be viewed as two synchronized P332 probes operating at up to 1.6 Gbps per lane. Note that currently, the P338 probe only supports the DSI standard.

The main purpose of the P338 probe is to provide synchronized video output on its two DPhy links. Frame timing is identical between the two links, but with each link having different source frame data. Options include acquiring source frame data from separate BMP files, or the left/right half or the even/odd pixels from a single BMP file.

On the other hand, both links do not always need to be active, and in single-interface modes, the P338 behaves almost identically to one P332 probe. A couple significant improvements in the P338 versus the P332 is that the firmware is updatable in the field and up to 4 KB of DUT response data can be acquired.

Interfaces

The P338 is considered to have two "interfaces", where an interface is analogous to the functionality of one P332 probe. Thus, an interface consists of two input connectors from the PG, internal processing elements, and one 4-lane DPhy link including a clock. The two interfaces are termed as the bottom (master) interface and the top (slave) interface. Descriptively, bottom and top refer to the physical locations of the probe connector outputs.

P338 Inputs from PG

Depending on the operating configuration of the probe via PGRemote, one module or two linked PG modules can be used for MIPI data storage and program output. When two PG modules are used this mode requires two PG modules connected with a Hydra cable and configured as a merged PG. This configuration essentially comprises a fully synchronous, double-width PG having 128 channels.

Generally, there is a one-to-one correspondence to the number of PG modules and how many DPhy links are active. However, there is one half-rate mode that allows both interfaces to be driven from a single PG. Also, one PG can drive either interface at full rate while the other is quiescent with all lanes in the LP11 state.

The PG inputs of the P338 are labeled P1A, P1C, P2A (P1B), and P2C (P1D). P1 and P2 refer to the PG module used to source the data input and A, B, C, and D refer to the which connector on the PG module is used. So, for example, P2C refers to a connection on the second PG module using the C connector. If a single PG module is used, the

connections in parentheses should be used (i.e. P1B and P1D). If two PG modules are used, the connections not in parentheses should be used (i.e. P2A and P2C).

P338 Outputs to MIPI

The MIPI outputs of the P338 are termed data lanes 0-7, clock1, and clock2. The bottom link consists of data lanes 0-3 and clock1. The top link consists of data lanes 4-7 and clock2 (within the link, data lanes 4-7 map to DPhy lanes 0-3 respectively).

5.2 PG Setup

The procedure for setting up the P338 probe depends on whether one or two PG modules are used to source data. Two linked PG modules are required to use the probe in dual-interface mode (i.e. both MIPI links) at full-rate (1.6 Gbps). Only one PG module is required to use the probe all other modes, including dual-interface mode at half-rate (800 Mbps) and single-interface mode at full-rate, where commands and video are sent to one MIPI link or the other but not both.

5.2.1 Two PG setup

If two PG modules are to be used, they must first be physically connected with a Hydra cable provided by the Moving Pixel Company, connected to the P338, and configured as a merged PG. To do this:

1. Choose one PG3A module to be the master and the other PG3A module to be the slave. In this document, the master module may also be referred to as PG1 or P1 and the slave as PG2 or P2.
2. Power off the modules.
3. Connect the Hydra cable to the modules:
    - Connect the single-connector end of the Hydra cable to the Hydra Out port of the master.
    - Connect one of the multiple-connector ends of the Hydra cable to the Hydra In port of the master.
    - Connect another of the multiple-connector ends of the Hydra cable to the Hydra In port of the slave.
4. Connect the PG modules to the P338:
    - Connect PG1, connector A to P338, connector P1A.
    - Connect PG1, connector C to P338, connector P1C.
    - Connect PG2, connector A to P338, connector P2A
    - Connect PG2, connector C to P338, connector P2C
5. Connect both master and slave to the host via USB (use USB hub if necessary).
6. Power on both modules.
7. Launch PGAppDotNet and select File->Choose PG… to open the Connect Dialog (note: this dialog will automatically open the first time PGAppDotNet is run).
8. Choose the appropriate serial numbers of the master and slave modules and click OK.
9. Click Yes to discard the current setup and data.

10. If successful, the System window should show a graphic with two pattern generator modules. The module LCD displays will read "P1" and "P2" more the master and slave modules respectively.

5.2.2 Single PG setup

If only one PG module is to be used, do the following:

1. Power off the module.
2. Connect the PG modules to the P338:
    - Connect PG1, connector A to P338, connector P1A.
    - Connect PG1, connector C to P338, connector P1C.
    - Connect PG1, connector B to P338, connector P1B
    - Connect PG2, connector D to P338, connector P1D
3. Connect the PG to the host via USB.
4. Power on the module.
5. Launch PGAppDotNet and select File->Choose PG… to open the Connect Dialog (note: this dialog will automatically open the first time PGAppDotNet is run).
6. Choose the module serial number of the master module and click OK.
7. Click Yes to discard the current setup and data.

5.3 Using PGRemote with the P338

PGRemote version 2.0 and greater supports P338 operation and is intended to continue to support the P331 and P332 probes as well. However, because of the significant changes necessary to support the P338, users of the P331 and P332 will not be immediately encouraged to upgrade without the understanding that version 2.x will be considered in Beta test for a period of time. Meanwhile, new features and fixes will be applied to both 1.x and 2.x versions.

Changes to the PGRemote user interface are minimal. They are summarized here and described futher in the next few sections.

- In the PG Configuration dialog, the user indicates the intended probe type and if the intended probe type is a P338, what operational configuration to use (i.e. dual-interface, single-interface modes).
- Also, in the PG Configuration dialog, voltage and delay controls have been added for data lanes 4-7 and Clock2.
- In the Timing Configuration dialog, if in a dual-interface mode, the user indicates how video data between the interfaces will be sourced (i.e. left/right image, even/odd pixel, etc).
- In the WriteMemory Configuration dialog, the user indicates the line length to use for binary (non-BMP) image files
- Finally, RPC commands have been added to support these new settings.

5.3.1 Setting the P338 Operating Configuration

The P338 has four operating configurations:

- Full-rate, Dual-interface – both output MIPI links are active and can output at bit rates up to 1.6 Gbps per lane. All commands are sent out with identical timing, on both interfaces. This includes all parameters that affect MIPI packets such as LaneCnt, DTMode, LPFreq, HSFreq, DPhy timing, frame timing, and any option settings. This configuration requires two merged PG modules.
- Half-rate, Dual-interface – similar to the full-rate, dual-interface configuration except that only one PG is used and the maximum bit rate is 800 Mbps per lane.
- Full-rate, Bottom-interface – only the bottom MIPI link is active and can output at bit rates up to 1.6 Gbps per lane. Essentially, the P338 in this configuration acts like a P332 probe connected to the bottom interface. Note that only one PG is required but this configuration will also work if two PGs are connected.
- Full-rate, Top-interface -- only the top interface is active and can output at bit rates up to 1.6 Gbps per lane. Essentially, the P338 in this configuration acts like a P332 probe connected to the top interface. Note that only one PG is required but this configuration will also work if two PGs are connected.

In either of the dual-interface configurations, DSI video-mode packets and WriteMemory commands may different image data between interfaces depending on the Dual-Interface Mode setting (Timing Configuration dialog). These modes are described further in the next section.

To set the operating configuration in the GUI, go to the PG Configuration dialog (PG Cfg… button in the main window), set the Probe Type to P338 and select the desired configuration in the P338 Config combo-box. Once the OK button is clicked in the dialog, the probe is reconfigured according to all the configuration settings.

In particular, the procedure for switching operating configuration of the P338 includes shutting down the clocks on both interfaces if they are active. Thus, after reconfiguration, all lanes of both links will be in LP11. Only once an initial command is sent, will the clock lane(s) re-enter HS mode (depending on the command and according to the same rules as for the P331/P332).

5.3.2 Setting the Dual-Interface Mode for Video

The dual-interface mode setting determines how video image data is mapped to active video packets and WriteMemory packets for each interface.[18] There are four dual-interface modes:

- Clone Image – Frame Timing parameters describe the video frame format for one interface. Identical image data is sent out each interface.
- Dual Image – Frame Timing parameters describe the video frame format for one interface. A special filename convention is used to import two image files, where

[18] Normally, Frame Timing parameters are not applicable to WriteMemory commands, but an exception is made here. The dual-interface mode setting is the used for WriteMemory commands in addition to video mode commands.

one image file is output on the top interface and one image is output on the bottom interface.

- Left/Right Image – Frame Timing parameters describe a combined video frame format for both interfaces. The bottom interface sends the left half of the image data and the top interface sends the right half of the image data.
- Even/Odd Pixel -- Frame Timing parameters describe a combined video frame format for both interfaces. The bottom interface sends even pixels of the image data and the top interface sends odd pixels of the image data. The first pixel of each line is considered to be an even pixel (pixel 0).

As a point of terminology, the Left/Right Image and Even/Odd Pixel modes are named "split-image" modes to distinguish them from Clone Image and Dual Image modes.

To configure the dual-interface mode, open the Timing Configuration dialog (Timing Cfg.. button in the main window) and select the desired option in the Dual-Interface Mode ComboBox at the bottom of the dialog. Note that this control will not be enabled unless the P338 configuration mode is set to one of the dual-interface modes (see previous section).

5.3.3 WriteMemory Configuration

The WriteMemoryStart and WriteMemoryContinue commands also support the dual-interface video modes. One complexity for PGRemote in sending a WriteMemory command is that the source data file can optionally be a binary data file (instead of a BMP file).

In a dual-interface mode, a WriteMemory command is considered to be sending image data and a binary data file has no corresponding line length information for parsing the file. The user provides this information in the WriteMemory Configuration dialog (Options->Configure WriteMemory… menu).

Note that if the file is a BMP, the line length is retrieved from the file itself. Also, even though the file is not a BMP file, the BMP Decode format is used to determine the pixel size of the binary input file.

WriteMemory partitioning is still supported in the dual-interface modes. However, one restriction for the split-image modes is that the WriteLen parameter must be a multiple of the video line length. This is so partitions can be made an integral number of lines and then split according to the mode. To do this, the partition length given by the user in the WriteMemory Configuration dialog will be quantized (rounded) to a multiple of the line length.

5.3.4 Specifying Image Filenames

Video-mode commands and WriteMemory commands are given an image filename as a parameter to the command. For video-mode commands, there already exists a special naming convention to allow for multiple frames in a sequence to be output. This naming

convention has been extended for video-mode and defined for WriteMemory commands to support dual-interface, dual-image mode.

Specifically, filenames used in dual-image mode require that the file names, not including extension, end in "A" or "B" (e.g. "ImageA.bmp" and "ImageB.bmp"). The "A" filename is specified in the command and the "B" filename is implicitly used. The "A" image is sent out the bottom interface and the "B" image is sent out the top interface.

For video-mode sequences, the protocol already exists to use an incrementing number at the end of files (e.g. "Image1.bmp", "Image2.bmp", "Image3.bmp"). For dual-interface, dual-image mode, filenames should have "A" or "B" following the incrementing number (e.g. "Image1A.bmp", "Image1B.bmp", "Image2A.bmp", "Image2B.bmp", etc.).

Finally, to allow for compatibility with non dual-image modes, the "A" and "B" dual-image sequences can be used in these modes. For example, sending the following command when P338 Config = "FULL-RATE TOP":

PktType = "Packed Pixel Stream (24-bit RGB 888)
DataFileName = "Image1A.bmp"
FrameCount = 3

The images sent out the top interface would be "Image1A.bmp", "Image2A.bmp" and "Image3A.bmp". Similarly, if the DataFileName were set to "Image1B.bmp", the images sent out the top interface would be be "Image1B.bmp", "Image2B.bmp" and "Image3B.bmp".

5.3.5 Read Commands

Due to the requirement that both interfaces are strictly synchronous, it is not currently possible to send a read command (or any command with BTA true) while in a dual-interface configuration. However, read commands are possible when in any single-interface configuration. Both the top and bottom interfaces can receive LPDT responses.

The following summarizes features and behavior of DUT response acquisition on the P338:

- The DUT response buffer on the P338 is 4 Kbytes.
- When the 4 Kbyte limit is reached during an acquisition, no further data is stored.
- Response data is read from the probe once the PG stops (and the response buffer in the probe is cleared). Thus, response data is not available while a command is looping.
- Multiple read responses can be acquired during looping or when sending a macro with multiple read commands.
- The DUT response display format in PGRemote has been modified to include a "BTA Entry" flag in the first byte of data received after each BTA to delimit multiple read responses.

As with the P331 and P332, the first byte displayed following a BTA represents the two bits of the DUT response, escape entry pattern of = 01b. In addition, the P338 appends the BTA Entry flag = 100h to this value. Thus, the first value expected after a read command is 101h. The second value expected is the LPDT entry code of 087h. Response data follows until the end of data or until another BTA Entry flag is seen, indicating a separate read response to follow.

PGRemote has been improved to display all acquired bytes of the DUT response. As in previous versions, the first several bytes of the response are displayed in the status area of the main window. Next to these bytes, there is now a small button with "…". Clicking this button brings up the DUT Response dialog which shows the entire response in a scrollable textbox.

In addition, this dialog provides a Save button to save the response data as text to a file. An RPC command is available for automating this process: SAVE_DUT_RESPONSE. Its single argument is the save filename. For example, in a script the command syntax would be:

SAVE_DUT_RESPONSE "c:\MIPI\responses\DUTResp1.txt"

5.3.6 Command Insertion

The P338 supports command insertion during video mode similarly to the P331/P332 probes. Insertion occurs on only one interface, top or bottom, depending on the current P338 configuration. In the "full-rate, top-interface" configuration, the top interface is used for command insertion. Otherwise, in all other configurations, including dual-interface settings, the bottom interface is used.

6 Frequently Asked Questions

This section provides answers to some of the frequently asked questions.

6.1 How do I implement a custom command?

To send any custom DSI or CSI command where the dataID is not defined in the standard, use the Custom Command packet type. Provide the custom DataID in the field provided and type in arbitrary argument bytes as a list of values separated by spaces in the Params[] field. You can append 'h' to make values hexadecimal. If the list contains two or less values, the command will be sent in the short packet format. Otherwise, it will be sent in the long packet format with the length field appropriately set.

If using RPC , use the CUSTOM_COMMAND RPC command, providing the custom DataID in the arg1 field. If using an RPC script, provide argument bytes as a list at the end of the command. If using RPC calls via the DLL, provide argument bytes in the "data" field of the MIPICmd DLL call.

6.2 How do I implement a custom DCS command?

To send a custom DCS short write command, you can select the 'DCS Short Write' as the PktType and 'Custom DCS Command' as the DCSCmdDesc. Then enter the custom DCS command in the DCSCmd field and zero, one, or two arbitrary bytes as parameters in the Params[] field searated by spaces.

To send either a DCS Read Request or a DCS Long Write, use the procedure for a generic custom command (above) putting the DCS command byte as the first byte in the argument list.

6.3 How do I send a command to a file, modify it with an in-line command, and send it?

1) Set up the desired packet type and its arguments to be sent in the left-side of PGRemote
2) Select the "Send Cmd To File" option in the File menu.and specify a file name to write the packet data to.
3) Open this file in a text editor and add the in-line commands you want before the #ASCII line
    a. e.g. add #HS_SOT A8 A8 A8 A8 or #LP_LPDT 86
4) Save the file
5) Back in PGRemote, select a PktType of "File Command" and browse for the text file name
6) Set up any configuration settings such as DTMode and frequencies.
7) Send the command.

6.4 How can I save my custom timing configurations?

Timing configurations are not saved with user command definitions. To permanently define new timing configurations, you should modify the text file in the application directory called Timing.Config. To change or add timing configurations, simply edit this file using a text editor. PGRemote does not provide the ability to save or modify configurations in this file using the GUI. The syntax of the file is documented in section 3.6.6.

Another way to set your timing configuration is to use an RPC script file. Often customers use this as method to initialize the entire state of PGRemote before sending a command or series of commands for testing. See next section.

6.5 How can I use PGRemote to respond to read requests as a slave device?

To do this, user's can build a macro that begins with a special command called "Wait For BTA" and is followed by whatever sequence of packets the user wants to send in response to being given the bus via BTA (e.g. this is often a read response packet to the known read packet that is to be sent by the DUT). The steps using the GUI would be:

- Click the Start Macro button
- Select "Wait For BTA" in the PktType drop-down
- Name the command if you want by typing in the DSI Cmd Name field
- Click the Send button to add it to the macro
- Select "Generic Short or Long Read Response" in the PktType drop-down
- Fill in the Params[] field and name the command by editing DSI Cmd Name if desired
- Click the Send button to add it to the macro
- Select "Bus Turn Around" in the PktType drop-down
- Click the Send button to add it to the macro
- Click the End Macro button, name it, and click OK.

Now you can send the macro and PGRemote will display "Waiting for BTA". However, after a couple seconds it will timeout. Used this way, you will want to check the option "Disable command timeout" (Options menu). Then, when the macro is sent, PGRemote will wait indefinitely until the DUT sends a BTA, at which point, the read response packet and subsequent return BTA will be sent (note that PGRemote will now be waiting for another return BTA as that is the semantics of the "Bus Turn Around" command). At this point, you can click "PGAbort" to reset PGRemote and the PG.

6.6 How can an RPC script file help with initialization?

A good way to set up and control almost all aspects of PGRemote is using one or more RPC scripts. In addition to their traditional use in sending a sequence of commands, you can use them to initialize all options, timing configuration, PG configuration, and DPhy configuration. Define a script file for each unique test configuration you need.

For example, for PG configuration, you could use the following commands

```
# START_EDIT_CONFIG
# SET_HS_LOW_VOLT 0 0.00
# SET_HS_HIGH_VOLT 0 0.45
# SET_HS_DELAY 0 00e-12
# SET_HS_DELAY 1 00e-12
# SET_HS_DELAY 2 00e-12
# SET_HS_DELAY 3 00e-12
# SET_HS_DELAY 4 500e-12
# SET_HS_FREQ 400e+6
# SET_LP_FREQ 20e+6
# SET_LANE_CNT 4
# END_EDIT_CONFIG
```

For frame timing, you could use:

```
# START_EDIT_CONFIG
# SET_TIMING_HSYNC 62
# SET_TIMING_HBPORCH 60
# SET_TIMING_HFPORCH 16
# SET_TIMING_HACTIVE 720
# SET_TIMING_VSYNC 6
# SET_TIMING_VBPORCH 31
# SET_TIMING_VFPORCH 8
# SET_TIMING_VACTIVE 480
# SET_TIMING_LINE_TIME 31.7778
# SET_TIMING_PIX_CLK 27.0000
# SET_TIMING_FRAME_RATE 59.9400

# SET_TIMING_ENABLE_CONTINUOUS_MODE 1
# SET_TIMING_ENABLE_DSI_BURST_MODE 0
# SET_TIMING_ENABLE_DSI_PULSE_MODE 1
# END_EDIT_CONFIG
```

And for DPhy timing configuration:

```
# START_EDIT_CONFIG
# SET_DPHY_PARAMETER DPHY_PARAM_HS_PREPARE 100
# SET_DPHY_PARAMETER DPHY_PARAM_HS_ZERO 100
# SET_DPHY_PARAMETER DPHY_PARAM_HS_EXIT 100
# SET_DPHY_PARAMETER DPHY_PARAM_HS_TRAIL 90
# SET_DPHY_PARAMETER DPHY_PARAM_CLK_PREPARE 70
# SET_DPHY_PARAMETER DPHY_PARAM_CLK_ZERO 260
# SET_DPHY_PARAMETER DPHY_PARAM_CLK_TRAIL 65
# SET_DPHY_PARAMETER DPHY_PARAM_CLK_PRE 45
```

```
# SET_DPHY_PARAMETER DPHY_PARAM_CLK_POST 320
# SET_DPHY_PARAMETER DPHY_PARAM_TA_GO 200
# SET_DPHY_PARAMETER DPHY_PARAM_WAKEUP 2000000
# END_EDIT_CONFIG
```

Finally, for PGRemote options:

```
# SET_OPTION OPT_CLOCK_ON_OFF_EACH_COMMAND 0
# SET_OPTION OPT_ENABLE_CMD_INSERTION 0
# SET_OPTION OPT_LOOP_NON_VIDEO_COMMANDS 1
# SET_OPTION OPT_ENABLE_EOT_PKTS 1
# SET_OPTION OPT_ENABLE_WM_PARTITIONING 1
# SET_OPTION OPT_DISABLE_CMD_TIMEOUT 0
# SET_OPTION OPT_RUN_PG_ON_EXTERNAL_EVENT 0
# SET_WM_PARTITION_LENGTH 40000
# SET_WM_PARTITION_INTERVAL 10
```

Note that there now is a shorthand way to generate this initialization script file. Simply do the following:

1. Select Script->"Start Recording"
2. Browse for the script file name to use
3. Select Script->"Write Current State"
4. Select Script->"Stop Recording"

6.7 How do I adjust the clock and data delays while a command or video is looping?

Here is the procedure:

1) Send looping command or video.
2) Bring up the PG Configuration dialog (click PG Cfg... button in main window)
3) Check the "Real-time Adjust Mode" check box
4) Click "No" in the message box that pops up asking whether it is "Okay to stop PG to update configuration".
5) Adjust clock or data delays as desired.
6) When done, click "Cancel" button in dialog
7) Click "No" in the message box that pops up asking whether it is "Okay to stop PG to update configuration".

6.8 How can I turn off the clock during LPDT commands?

There is a way to turn off the clock during an LP command for a specific test. That is to build a macro with the "Clock Off" command preceding the LP command. An RPC script to do this is as follows:

```
# START_MACRO
# SEND_MIPI_CMD CLOCK_OFF 0 0 DT_HS 0 0 0 0 "" NULL
# SEND_MIPI_CMD DCS_SHORT_WRITE EXIT_SLEEP_MODE 0 DT_LP 0 0 0 0 ""
NULL
# SEND_MACRO
```

Also, a related issue is to ensure the clock is off during LP blanking during video mode. This can be achieved by checking the checkbox "Turn clock off during LP blanking" in the New Video Mode configuration fields of the Timing configuration dialog.

6.9 How can I send multiple video frames and step through them one by one using an external event (DSI video only)?

To do this, set up to send multiple video frames as normal (i.e. set FrameCount greater than one and provide multiple BMP filenames with a proper incrementing number sequence). In the timing configuration, both "New Video Mode" and continuous video must be selected. Then, in the Options menu, check "Run PG on external event".

In this case, an external event line connected to the Event0 line of the P300 Inputs probe to the PG controls switching to the next video frame in the sequence. Whenever the event line pulses high, video steps to the next frame. After sending the video command, the event must occur to start playing the first frame of video. This frame will loop until either PGAbort is pressed (which will end the video playback as usual) or the event signal occurs. When the event signal occurs, the second frame will loop until either PGAbort is pressed or the event signal occurs again. And so on. Once the last frame has played, the first frame will play again.

6.10 How can I efficiently send long LP-only commands?

To answer this question, some details about how the PG is used to store and send MIPI commands. Commands are built and stored in the PG before they are sent out using the P331/P332 probe. The PG memory contains about 32 million vectors of memory. This represents the limit of the size of any command, video sequence, or macro that can be sent at one time.

PG vectors are effectively played out at the 2 * HS frequency, even for LP commands. This means that multiple vectors are used for each LP bit and thus can require significant memory depending on the ratio of HS frequency to LP frequency. Because LP bits are one-spaced hot encoded (requiring 2 pulses per LP bit) the exact equation is:

Vectors per LP bit = (4 * HSFreq / LPFreq).

The most important piece of information to take away from this is that when sending large LP-only commands, you should set the LP frequency as high as possible or set the HS frequency as low as possible. In the case of LP-only commands, PGRemote actually allows HSFreq to be set equal to the LPFreq (even though the minimum HS frequency in DPhy is 80 MHz). This allows the most efficient memory usage in the PG and drastically increases the download time for large LP commands (e.g. WriteMemory).

6.11 What do I do when PGRemote displays "Unable to stop PG"?

"Unable to stop PG" means that PGAppDotNet is not responding to requests from PGRemote to stop the PG . This means that PGAppDotNet is in a confused state and should be shut down. These kinds of strange errors should not happen very often, but when they do, it is best to shut everything down (close both PGRemote and PGAppDotNet) and start anew. Make sure there are not other copies of the software running. And, if all else fails, it may be best to reboot the computer.

When starting up again, relaunch PGAppDotNet, connect to your PG (if it doesn't do so automatically), then bring up PGRemote. You should not have any connection or initialization problems.

Appendix A RPCErrs Class Reference

```
namespace PGRemoteRPC
{

    public class RPCBaseErr
    {
        public const int RPC_FAILED = -100000;
        public const int NOT_LICENSED = -100001;
        public const int SERVER_BUSY = -100002;
        public const int NO_CALLBACK = -100003;
        public const int NO_SERVER_CONNECTION = -10004;
    }

    public class RPCErrs : RPCBaseErr
    {
        public const int FAIL = -1;
        public const int INVALID_STATE = -2;
        public const int UNKNOWN_CMD = -3;
        public const int INVALID_ARG = -4;
        public const int BAKPLANE_NOT_PRESENT = -5;
        public const int NOT_LICENSED = -6;
        public const int ARGTYPE_MISMATCH = -7;
        public const int USB_OPEN_FAILED = -100;
        public const int DEVICE_NOT_PRESENT = -101;
        public const int USB_TIMEOUT = -102;
        public const int MAX_LEN_EXCEEDED = -103;
        public const int CANT_OPEN_FILE = -104;
        public const int FILE_DOESNT_EXIST = -105;
        public const int FILE_TOO_SHORT = -106;
        public const int IO_ERROR = -107;
        public const int INVALID_PARAM = -108;
        public const int BAD_RBF_FILE_LEN = -109;
        public const int BAD_MAGIC_NUMBER = -110;
        public const int DDRBW_FIFO_OVERFLOW = -111;
        public const int DDRBW_DID_NOT_COMPLETE = -112;
        public const int VERIFY_FAILED = -113;
        public const int ABORT = -114;
        public const int TIMEOUT = -115;
        public const int CANT_ACQUIRE_SEMA = -116;
        public const int NO_PROGSTATE_DEFINED = -117;
        public const int BAD_SERIAL_NUMBER = -118;
        public const int BAD_FIRMWARE_FILE_LEN = -119;
        public const int BAD_SECURITY_FILE = -120;
        public const int INVALID_FIRMWARE_FILE = -121;
        public const int FLASH_VERIFY_ERROR = -122;
        public const int CAPT_STATE_MISMATCH = -123;
        public const int NO_PG_CONNECTION = -124;
        public const int REMOTE_PG_CALL_FAILED = -125;
        public const int BAD_FRAME_PARAMETERS = -126;
        public const int BAD_LINE_TIME = -127;
        public const int INVALID_SYNC_LINE_CNT = -128;
        public const int INVALID_ACTIVE_PIX_CNT = -129;
```

```
public const int INVALID_HEIGHT_OR_WIDTH = -130;
public const int CANT_CONFIGURE_P375 = -131;
public const int PG_IS_NOT_IDLE = -132;
public const int INSUFFICIENT_PG_MEMORY = -133;
public const int THREAD_IS_BUSY = -134;
public const int INVALID_TIMING_PARAM = -136;
public const int INVALID_HSYNC_PARAM = -137;
public const int INVALID_HFRONTPORCH_PARAM = -138;
public const int INVALID_HBACKPORCH_PARAM = -139;
public const int AGGREGATE_HS_PKT_LANE_MISMATCH = -140;
public const int NO_BLANKING_ADJ = -141;
public const int HSA_TOO_SHORT = -142;
public const int HBP_TOO_SHORT = -143;
public const int HFP_TOO_SHORT = -144;
public const int MAX_PAYLOAD_EXCEEDED = -145;
public const int TEST_DID_NOT_COMPLETE = -146;
public const int INVALID_PROBE_TYPE = -147;
public const int SESSION_EXPIRED = -148;
public const int UNEXPECTED_PROBE_TYPE = -149;
public const int INVALID_DELAY_SETTING = -150;
public const int INVALID_EEPROM_VALUE = -151;
public const int INVALID_VBACKPORCH_PARAM = -152;
public const int BAD_HS_MODE = -153;
public const int BAD_LP_REPL = -154;
public const int SEG_UNALIGNED = -155;
public const int INPUTS_PROBE_NOT_PRESENT = -156;
public const int INPUT_CLOCK_MISMATCH = -157;
public const int BLOCK_LEN_IS_NOT_EVEN = -158;
public const int P331_CLOCK_ISNT_RUNNING = -159;
public const int UNEXPECTED_CMD_PHASE = -160;
public const int INSERTION_CODE_NOT_FOUND = -161;
public const int INSERTION_CMD_TOO_LONG = -162;
public const int BTA_TIMEOUT = -163;
public const int NO_RESPONSE_FROM_PROBE = -164;
public const int NO_EXTERNAL_CLOCK = -165;
public const int CANT_LOCK_HS_PLL = -166;
public const int BAD_EXTERNAL_CLOCK_FREQ = -167;
public const int MISSING_PROBE_CONNECTION = -168;
public const int INVALID_HS_BITRATE = -169;
public const int CANT_RESTART = -170;
public const int BAD_P331_CLOCK_STATE = -171;
public const int PARSE_ERR = -172;
public const int UNKNOWN_MIPI_CMD = -173;
public const int NEED_START_EDIT_CMD = -174;
public const int CONTROL_IS_DISABLED = -175;
public const int UNEXPECTED_ARG_TYPE = -176;
public const int PAYLOAD_TOO_SHORT = -177;
public const int CMD_STANDARD_MISMATCH = -178;
public const int UNRECOGNIZED_COMMAND = -179;
public const int UNKNOWN_DCS_CMD = -180;
public const int CANT_PARTITION_FILE_CMD = -181;
public const int LP_DELAY_TOO_LONG = -182;
public const int INSERTION_BLOCK_TOO_LONG = -183;
public const int CANT_CREATE_PG_BLOCK = -184;
public const int PG_IS_NOT_ARMED = -185;
public const int PG_IS_NOT_RUNNING = -186;
public const int PG_IS_NOT_WAITING = -187;
```

```
        public const int PG_IS_NOT_IN_EXPECTED_STATE = -188;
        public const int DUAL_VIDEO_SYNC_NOT_SUPPORTED = -189;
        public const int USE_EXT_CLK_ON_P331_AS_REF_NOT_SUPPORTED = -190;
        public const int TOO_FEW_TOKENS = -191;
        public const int VALUE_OUT_OF_RANGE = -192;
        public const int OBJECT_NOT_FOUND = -193;
        public const int USER_CANCELED = -194;
        public const int SEGMENT_NOT_INTEGRAL_HALF_BYTES = -195;
        public const int SEGMENT_NOT_INTEGRAL_HALF_BYTE_CLOCKS = -196;
        public const int CANT_IMPL_LP11_NO_LANE0_END = -197;
        public const int CANT_IMPL_LP11_NEED_VARIABLE_DPHY_TIMING = -198;
        public const int BLANKING_SEGMENT_TOO_SHORT = -199;
        public const int SEGMENT_NOT_INTEGRAL_BYTES = -200;
        public const int SEGMENT_NOT_INTEGRAL_BYTE_CLOCKS = -201;
        public const int VTOTAL_MUST_BE_EVEN = -202;
        public const int INVALID_BLANKING_BYTES = -203;
        public const int SEGMENT_NOT_EVEN_BYTE_CLOCKS = -204;
        public const int BLANKING_SEGMENT_TOO_LONG = -205;
        public const int INVALID_VACTIVE_PARAM = -206;
        public const int INVALID_VFRONTPORCH_PARAM = -207;

        public const int PORT_IS_UNAVAILABLE = -300;
        public const int CANT_FIND_PGAPPDOTNET_INSTALL_DIR = -301;
        public const int UNSUPPORTED_PGAPPDOTNET_CMD = -302;
        public const int UNSUPPORTED_PGAPPDOTNET_VERSION = -303;
        public const int P338_NOT_IN_DUAL_ACTIVE_INTERFACE = -304;
        public const int CANT_PARSE_VALUE = -305;

        public const int LOCAL_FAILURE = -5000;
    }
}
```

Appendix B RPCCmds Class Reference

```
namespace PGRemoteRPC
{

    public class RPCCmds : RPCBaseCmds
    {
        // see Table 4 for MipiCmd arguments
        public const int SEND_MIPI_CMD = 1;

        // arg1 = byte array of length 3
        public const int COMPUTE_ECC = 2;

        // arg1 = byte array
        public const int COMPUTE_CRC = 3;

        // see Table 4 and 5 for ImpairedMipiCmd arguments
        public const int SEND_IMPAIRED_MIPI_CMD = 4;

        ////////////////////////////////////////////////
        // PG & probe configuration commands
        //
        // Bracket any series of PG/probe configuration calls with
        // START_EDIT_CONFIG and END_EDIT_CONFIG

        ////////////////////////////////////////////////

        // no arguments
        public const int START_EDIT_CONFIG = 0x20d;

        // no arguments
        // initializes probe and PG with new configuration settings
        public const int END_EDIT_CONFIG = 0x20e;

        // arg1 = 0 (disable), 1 (enable)
        public const int ENABLE_REAL_TIME_ADJUST = 0x20f;
        public const int ENABLE_AUTO_SET_CLOCK_DELAY = 0x232;
        public const int ENABLE_FORCE_CLOCK_PATTERN = 0x233;

        // arg1 = mode (DT_HS, DT_LP)
        public const int SET_DT_MODE = 0x210;

        // arg1 = LP_FREQ (float, Hz)
        // (range: 10 KHz - 30 MHz)
        // Note: will be quantized based on HS_FREQ setting
        // (see PGemote manual)
        public const int SET_LP_FREQ = 0x211;

        // arg1 = HS_FREQ (float, Hz)
        // (range P331, internal clock mode: 500 KHz - 500 MHz)
        // (range P331, external clock mode: 12.5 MHz - 500 MHz)
        // (range P338, internal clock mode: 750 MHz)
        // Note: will cause LP_FREQ to be quantized
```

```
// (see PGRemote manual)
public const int SET_HS_FREQ = 0x212;

// arg1 = lane count (1-4)
public const int SET_LANE_CNT = 0x213;

// arg1 = probe type (PT_P331, PT_P332, PT_P338)
public const int SET_PROBE_TYPE = 0x214;

// arg1 = 0 (disable), 1 (enable)
public const int ENABLE_ALL_HS_VOLT_COMMON = 0x215;

// arg1 = 0 (disable), 1 (enable)
// Note: P331 cannot disable this setting
public const int ENABLE_ALL_LP_VOLT_COMMON = 0x216;

// arg1 = channel (0-4) for P331/P332 (0-9) for P338
//        chan 0-3 are data lanes 0-3
//        chan 4 is clock lane
//        chan 5-8 are data lanes 4-7 of P338
//        chan 9 is clock2 lane of P338
// arg2 = voltage (float, volts)
// (range P331: -0.6 - 1.2)
public const int SET_HS_LOW_VOLT = 0x217;
public const int SET_HS_HIGH_VOLT = 0x218;

// arg1 = channel (0-4) for P331/P332 (0-9) for P338
//        chan 0-3 are data lanes 0-3
//        chan 4 is clock lane
//        chan 5-8 are data lanes 4-7 of P338
//        chan 9 is clock2 lane of P338
// arg2 = voltage (float, volts)
// (range P331: must be 0.0)
public const int SET_LP_LOW_VOLT = 0x219;

// arg1 = channel (0-4) for P331/P332 (0-9) for P338
//        chan 0-3 are data lanes 0-3
//        chan 4 is clock lane
//        chan 5-8 are data lanes 4-7 of P338
//        chan 9 is clock2 lane of P338
// arg2 = voltage (float, volts)
// (range P331: 0.45 - 3.6)
public const int SET_LP_HIGH_VOLT = 0x21a;


// arg1 = channel (0-4) for P331/P332 (0-9) for P338
//        chan 0-3 are data lanes 0-3
//        chan 4 is clock lane
//        chan 5-8 are data lanes 4-7 of P338
//        chan 9 is clock2 lane of P338
// arg2 = delay (float, sec)
// (range: 0 - 2400 ps)
public const int SET_HS_DELAY = 0x21b;
public const int SET_LP_DELAY = 0x21c;

// arg1 = voltage (float, volts)
// (range P331: 0.0 - 1.0)
```

```
public const int SET_LP_LOW_CONT_THRESH = 0x21d;
public const int SET_LP_HIGH_CONT_THRESH = 0x21e;

// arg1 = time (float, sec)
// (range P331: not supported)
public const int SET_BTA_WAIT_TIME = 0x21f;

// arg1 = 0 (disable), 1 (enable)
public const int ENABLE_EXTERNAL_CLOCK = 0x220;

// arg1 = P338 config (PGRemoteForP338 only)
// (FULL_RATE_DUAL_INTERFACE, HALF_RATE_DUAL_INTERFACE,
// FULL_RATE_BOTTOM_INTERFACE, FULL_RATE_TOP_INTERFACE)
public const int SET_P338_CONFIG = 0x400;

// arg1 = P300 event thresh (float, V) (PGRemoteForP338 only)
// (range -2.0 - 2.5 V)
public const int SET_PG_EVENT_THRESHOLD = 0x401;

// arg1: (float, seconds) (PGRemoteForP338 only)
// (max: 80 ms, quantized to 5 ns increments)
public const int SET_PROBE_TRIG_DURATION = 0x402;

// arg1 = 0 (disable), 1 (enable)
// enables using the P338 probe trigger as a frame
// advance signal during video mode
public const int ENABLE_AUTO_FRAME_ADVANCE = 0x403;

// arg1: (int, frame count, range: 1 - 1000)
// sets the number of frames between trigger pulses when
// used for auto-frame advance
public const int SET_AUTO_FRAME_ADVANCE_COUNT = 0x404;

// Asserts TrigOut on the P338 probe (P338-Only)
// (also duplicated on GPO0 back-panel signal)
//
// NOTE: this is for immediate probe trigger control;
// (use ASSERT_PROBE_TRIG to set TrigOut from MIPI stream)
// arg1 = trigger command: 0 (low), 1 (high),
//                         2 (toggle), 3 (pulse)
// Note: Toggle command starts from current voltage
//       (i.e. may be low->high or high->low)
// Note: Pulse command starts from current voltage
//       (i.e. may be low->high->low or high->low->high)
// Use SET_PROBE_TRIG_DURATION oo set the pulse width
public const int FORCE_PROBE_TRIG = 0x405;

// Sets the link delay for the given interface (P338-only)
// arg1 = 0 (bottom interface), 1 (top interface)
// arg2 = delay (float, seconds)
//        delay range is 0 to 188 * HS Byte Clock period)
//        delay is quantized to HS Byte Clock period multipl
public const int SET_LINK_DELAY = 0x406;

// Sets the interface(s) to use for command insertion
// when in dual-link mode (P338-only)
// arg1 = 0 (bottom interface), 1 (top interface),
```

```
        //        2 (both top and bottom interface)
        public const int SET_DUAL_LINK_INSERT_IFACE = 0x407;

        ////////////////////////////////////////////////
        // timing configuration commands
        ////////////////////////////////////////////////

        // arg1 = pixel count
        public const int SET_TIMING_HSYNC = 0x221;
        public const int SET_TIMING_HBPORCH = 0x222;
        public const int SET_TIMING_HFPORCH = 0x223;
        public const int SET_TIMING_HACTIVE = 0x224;

        // arg1 = line count
        public const int SET_TIMING_VSYNC = 0x225;
        public const int SET_TIMING_VBPORCH = 0x226;
        public const int SET_TIMING_VFPORCH = 0x227;
        public const int SET_TIMING_VACTIVE = 0x228;

        // arg1 = line time (float, sec)
        public const int SET_TIMING_LINE_TIME = 0x229;

        // arg1 = pixel clock frequency (float, Hz)
        public const int SET_TIMING_PIX_CLK = 0x22a;

        // arg1 = frame rate (float, Hz)
        public const int SET_TIMING_FRAME_RATE = 0x22b;

        // arg1 = 0 (disable), 1 (enable)
        public const int SET_TIMING_ENABLE_CONTINUOUS_MODE = 0x22c;
        public const int SET_TIMING_ENABLE_DSI_PULSE_MODE = 0x22d;
        public const int SET_TIMING_ENABLE_DSI_BURST_MODE = 0x22e;
        public const int SET_TIMING_ENABLE_CSI_LSLE_MODE = 0x22f;
        public const int SET_TIMING_ENABLE_CSI_FRAME_NUMBERING = 0x230;
        public const int SET_TIMING_ENABLE_CSI_LINE_NUMBERING = 0x231;

        // arg1 = 0 (disable), 1 (enable)
        public const int SET_TIMING_USE_NEW_VIDEO_MODE = 0x300;
        public const int SET_TIMING_ALLOW_VARIABLE_DPHY_TIMING = 0x301;
        public const int SET_TIMING_TURN_CLOCK_OFF_DURING_LP_BLANKING =
0x302;

        // arg1 = blanking mode
        // (AUTO_BLANK_MODE, HS_BLANK_MODE, LP11_BLANK_MODE)
        public const int SET_TIMING_HSYNC_BLANKING_MODE = 0x303;
        public const int SET_TIMING_HBPORCH_BLANKING_MODE = 0x304;
        public const int SET_TIMING_HFPORCH_BLANKING_MODE = 0x305;
        public const int SET_TIMING_VERTICAL_BLANKING_MODE = 0x306;
        public const int SET_TIMING_TOP_FIELD_FIRST = 0x308;

        // arg1 = dual-interface mode (P338 probe only)
        // (DI_CLONE_INTERFACE, DI_DUAL_IMAGE,
        // DI_LEFT_RIGHT_IMAGE, DI_EVEN_ODD_PIXEL)
        public const int SET_TIMING_DUAL_INTERFACE_MODE = 0x307;

        // arg1 = 0 (disable), 1 (enable)
        public const int SET_TIMING_TOP_FIELD_FIRST = 0x308;
```

```
// arg1 = 0 (disable), 1 (enable) (P338 probe only)
public const int SET_TIMINE_ENABLE_OVERLAP = 0x309;

// arg1 = overlap setting (in pixels) (P338 probe only)
public const int SET_TIMING_OVERLAP = 0x30a;



////////////////////////////////////////////////
// other commands
////////////////////////////////////////////////

// arg1 = hostname (string)
public const int PG_CONNECT = 0x240;

// no arguments
public const int PG_DISCONNECT = 0x241;

// arg1 = standard (STD_DSI, STD_CSI)
public const int SET_MIPI_STANDARD = 0x242;

// arg1 = option (see OPT_XXX defines))
// arg2 = 0 (disable), 1 (enable)
public const int SET_OPTION = 0x243;

// no arguments
public const int PG_ABORT = 0x244;
public const int PG_RESTART = 0x245;

// no arguments; returns 0 (not connected),
// 1 (connected), <0 (error)
public const int IS_PG_CONNECTED = 0x246;

// no arguments; returns 0 (not running),
// 1 (running), <0 (error)
public const int IS_PG_RUNNING = 0x247;

// no arguments; returns 0 (not running),
// 1 (running), <0 (error)
public const int IS_LANE_CLK_RUNNING = 0x248;

// no arguments; returns contention mask, <0 (error)
// mask consists of four bits {CTN_LP0L, CTN_LP1L,
// CTN_LP0H, CTN_LP1H}
public const int GET_LP_CONTENTION = 0x249;

// no arguments, resets contention mask in hardware
public const int RESET_LP_CONTENTION = 0x24a;

// no arguments; returns MIPI probe type connected
// to PG connector A,
// 0 for no probe or non-MIPI probe, <0 for error
// (note this is not necessarily the same as the value set with
// the SET_PROBE_TYPE command, which is the intended probe type
// for PGRemote to use)
```

```
public const int GET_PG_PROBE_TYPE = 0x24b;

// no arguments; puts PGRemote in macro mode
// (discarding previous macro)
// any (non-video) MIPI_CMDs sent will append to current macro
public const int START_MACRO = 0x24c;

// no arguments; ends and sends the current macro
// repeated calls will send the macro repeatedly
public const int SEND_MACRO = 0x24d;

// arg1 = parameter (DPHY_PARAM_* definition)
// arg2 = value (int, ns)
//   {except for USE_SYSTEM_COMPUTED_TIMING where
//   arg2 = enable (int, 0 or 1)}
public const int SET_DPHY_PARAMETER = 0x24e;

// arg1 = parameter (DPHY_PARAM_* definition)
// gets current DPhy parameter setting
public const int GET_DPHY_PARAMETER = 0x24f;

// arg1 = parameter (DPHY_PARAM_* definition)
// gets actual quantized DPhy parameter value used by PG
public const int GET_REAL_DPHY_PARAMETER = 0x250;

// arg1 = user message string enclosed in double-quotes
// displays message box to user with OK & CANCEL button
// OK returns no error
// CANCEL returns ABORT which will terminte RPC script
public const int USER_WAIT = 0x251;

// arg1 = maximum number of bytes per DCS WriteMemory partition
public const int SET_WM_PARTITION_LENGTH = 0x252;

// arg1 = LP11 time between DCS WriteMemory partitions
// (float, sec)
public const int SET_WM_PARTITION_INTERVAL = 0x253;

// arg1 = DCS WriteMemory BMP decode format
// (FMT_RGB565, FMT_RGB666_18, FMT_RGB666_24, FMT_RGB888_24)
public const int SET_WM_BMP_DECODE_FORMAT = 0x254;

// no arguments, (use PGRemoteQuery call in PGRemoteRPCClient)
public const int GET_DUT_RESPONSE = 0x255;

// arg1 = time to delay before playing video (float, sec)
// used for dual-video synchronization
public const int SET_DUAL_VIDEO_DELAY_INTERVAL = 0x256;

// arg1 = line length in pixels used to decode binary files
// when used with the P338 in dual-interface, split-image modes
public const int SET_BINARY_FILE_LINE_LENGTH = 0x257;

// Saves DUT response from last read command to text file
// arg1 = fileName: name of file to save DUT response
// arg2 = appendFlag (int): set to non-zero to append to file
// arg3 = commentString: comment to save with response;
```

```
//        set to "" for no comment
public const int SAVE_DUT_RESPONSE = 0x258;

// Maps logical lane data to physical output lanes
//
// arg1 = lane mapping integer
// (set to -1 to disable lane mapping)
// set lane mapping as follows:
// laneMap[3..0] source data lane to use for phy lane 0
// laneMap[7..4] source data lane to use for phy lane 1
// laneMap[11..8] source data lane to use for phy lane 2
// laneMap[15..12] source data lane to use for phy lane 3
// laneMap[19..16] source data lane to use for phy clock lane
// Use lane = 4 for clock lane
// For example, unity mapping is: 43210h
//
// Note: this function maps both interfaces of the P338 to
// the given configuration.
public const int SET_LANE_MAP = 0x259;

// no arguments, returns eventState[7..0] (integer)
// (Event0 => bit0, …, Event7 => bit7)
public const int GET_PG_EVENT_STATE = 0x260;

// arg1: fileName: filename to save PG event state (as int)
// arg2: appendFlag (int): set to non-zero to append to file
// arg3: commentString: comment to save with response;
//       set to "" for no comment
public const int SAVE_PG_EVENT_STATE = 0x261;

// no arguments, returns number of named Cmds defined (integer)
public const int GET_NAMED_CMD_COUNT = 0x262;

// arg1: Cmd index (integer)
// returns the Cmd name (string) associated with the given
// Cmd index
public const int GET_NAMED_CMD_NAME = 0x263;

// arg1: Cmd name (string)
// returns the RPC string associated with the given Cmd name
public const int GET_RPC_CMD_STRING_FROM_NAME = 0x264;

// arg1: Cmd index (integer)
// returns the RPC string associated with the given Cmd index
public const int GET_RPC_CMD_STRING_FROM_INDEX = 0x265;

// arg1: RPC command string (string)
// executes the given RPC command string
// the string may have multiple RPC commands separated by
// <CR><LF> or <CR>
// (all rules for RPC script files are used)
public const int EXECUTE_RPC_CMD_STRING = 0x266;

// arg1: Cmd name (string)
// sends the (already defined) named command given its
// Cmd name, as if sent from the GUI
public const int SEND_NAMED_CMD_FROM_NAME = 0x267;
```

```
// arg1: Cmd index (integer)
// sends the (already defined) named command given its
// Cmd index, as if sent from the GUI
public const int SEND_NAMED_CMD_FROM_INDEX = 0x268;

// arg1: buffer number, (integer, 0-3)
// arg2: dataType specifying video format that will
//       be used with this frame (integer)
// arg3: file name (string)
// Loads the frame from the given file, converts to
// the given video format, and stores
// into the given frame buffer number
// (use DEALLOC_FRAME to deallocate when done)
// (PGRemoteForP338 only)
public const int LOAD_FRAME = 0x269;

// arg1: buffer number, (integer, 0-3)
// Deallocates the frame stored in the given frame buffer
// (PGRemoteForP338 only)
public const int DEALLOC_FRAME = 0x26a;

// no arguments; ends and measures the macro length
// returns the number of bytes required to implement
// the current macro (macro is discarded after measuring)
// (PGRemoteForP338 only)
public const int MEASURE_MACRO = 0x26b;

////////////////////////////////////////////////
// GUI Commands (PGRemoteForP338 Only)
////////////////////////////////////////////////

// Loads the button and command configuration
// from the given configuration file.
// arg1: configuration file name (string)
public const int LOAD_CONFIG = 0x500;

// Saves the current button and command configuration
// to the given configuration file.
// arg1: configuration file name (string)
public const int SAVE_CONFIG = 0x501;

// Assigns command to a button in the application
// (Will replace an existing button assignment,
// use "" to unassign)
// (use button page = -1 for all pages)
// (use button number = -1 for all buttons on page)
// arg1: button page (0-3, int)
// arg2: button number in page (0-29, int)
// arg3: command name (string)
// arg4: tooltip (string)
public const int ASSIGN_BUTTON = 0x502;

// Create a named macro in the application
// Use at end of macro definition instead of SEND_MACRO
// (Will delete an existing macro by the same name)
// arg1: macro name (string)
```

```
        public const int ADD_MACRO = 0x503;

        // Create a named command in the application
        // (Will delete an existing command by the same name)
        // (If sent while in START_MACRO / ADD_MACRO,
        //  only names the component command being added to macro)
        // Uses same arguments as SEND_MIPI_CMD with one
        // additional argument (first argument)
        // First argument: command name (string)
        public const int ADD_MIPI_CMD = 0x504;

        // Deletes a named command in the application
        // arg1: command name (string)
        public const int DELETE_CMD = 0x505;

        // Deletes all named commands in the application
        // no arguments
        public const int DELETE_ALL_CMDS = 0x506;


    }
}
```

Appendix C RPCDefs Class Reference

```
namespace PGRemoteRPC
{
    public class RPCDefs
    {
        // DT modes
        public const int DT_DEFAULT = 0;
        public const int DT_LP = 1;
        public const int DT_HS = 2;

        // probe types
        public const int PT_P331 = 15;
        public const int PT_P332 = 17;
        public const int PT_P338 = 18;

        // MIPI standards
        public const int STD_DSI = 0;
        public const int STD_CSI = 1;
        public const int STD_GENERIC = 2;

        // options
        public const int OPT_CLOCK_ON_OFF_EACH_COMMAND = 0;
        public const int OPT_ENABLE_CMD_INSERTION = 1;
        public const int OPT_LOOP_NON_VIDEO_COMMANDS = 2;
        public const int OPT_ENABLE_EOT_PKTS = 3;
        public const int OPT_ENABLE_WM_PARTITIONING = 4;
        public const int OPT_DISABLE_CMD_TIMEOUT = 5;
        public const int OPT_SEND_SINGLE_PKT_PER_HS_BURST = 6;
        public const int OPT_ONLY_LP11_VIDEO_BLANKING = 7;
        public const int OPT_RUN_PG_ON_EXTERNAL_EVENT = 8;
        public const int OPT_USE_EXT_CLOCK_INPUT_ON_P331_
                                    AS_10MHZ_REFERENCE = 9;
        public const int OPT_ENABLE_DUAL_VIDEO_SYNCHRONIZATION = 10;
        public const int OPT_APPLY_EOT_PKTS_TO_LPDT = 11;

        // DCS Write Memory BMP decode formats
        public const int FMT_RGB565_16 = 1;
        public const int FMT_RGB666_18 = 2;
        public const int FMT_RGB666_24 = 3;
        public const int FMT_RGB888_24 = 4;
        public const int FMT_YCBCR_420_12 = 11;
        public const int FMT_YCBCR_422_16 = 13;
        public const int FMT_RGB101010_30 = 23;
        public const int FMT_RGB121212_36 = 24;
        public const int FMT_YCBCR_422_LOOSE_20 = 25;
        public const int FMT_YCBCR_422_24 = 26;

        // contention mask bits
        public const int CTN_LP0L = (1 << 0);
        public const int CTN_LP1L = (1 << 1);
        public const int CTN_LP0H = (1 << 2);
        public const int CTN_LP1H = (1 << 3);
```

```
// DPhy parameters
public const int DPHY_PARAM_HS_PREPARE = 0;
public const int DPHY_PARAM_HS_ZERO = 1;
public const int DPHY_PARAM_HS_EXIT = 2;
public const int DPHY_PARAM_HS_TRAIL = 3;
public const int DPHY_PARAM_CLK_PREPARE = 4;
public const int DPHY_PARAM_CLK_ZERO = 5;
public const int DPHY_PARAM_CLK_TRAIL = 6;
public const int DPHY_PARAM_CLK_PRE = 7;
public const int DPHY_PARAM_CLK_POST = 8;
public const int DPHY_PARAM_TA_GO = 9;
public const int DPHY_PARAM_WAKEUP = 10;
public const int DPHY_PARAM_USE_SYSTEM_COMPUTED_TIMING = 30;

// New DSI blanking modes
public const int AUTO_BLANK_MODE = 0;
public const int LP11_BLANK_MODE = 1;
public const int HS_BLANK_MODE = 2;

// dual-interface modes (P338 only)
// (replicated from DSIFrameTiming class)
public const int DI_CLONE_INTERFACE = 0;
public const int DI_DUAL_IMAGE = 1;
public const int DI_LEFT_RIGHT_IMAGE = 2;
public const int DI_EVEN_ODD_PIXEL = 3;

// P338 configuration modes (P338 only)
public const int FULL_RATE_DUAL_INTERFACE = 0;
public const int HALF_RATE_DUAL_INTERFACE = 1;
public const int FULL_RATE_BOTTOM_INTERFACE = 2;
public const int FULL_RATE_TOP_INTERFACE = 3;

////////////////////////////////////////////////
// DSI commands
//
// Note: the command values defined here are applicable
// when using RPC script files and match the command codes
// in the DSI specification.  On the other hand, when
// using these command codes in RPC calls via the
// PGRemoteRPCClient DLL, all DSI commands will have 0x400
// and all CSI commands will have 0x500 added to them
// (to make them unique across standards).
////////////////////////////////////////////////

// 0 argument commands
public const int VSYNC_START = 0x1;
public const int VSYNC_END = 0x11;
public const int HSYNC_START = 0x21;
public const int HSYNC_END = 0x31;
public const int EOT_PKT = 0x08;
public const int COLOR_MODE_OFF = 0x02;
public const int COLOR_MODE_ON = 0x12;
public const int SHUT_DOWN_PERIPHERAL = 0x22;
public const int TURN_ON_PERIPHERAL = 0x32;
public const int DCS_READ = 0x6;
```

```
// arg1 = ParamCnt (0, 1, 2)
// arg2 = Parameter byte 1 (if necessary)
// arg3 = Parameter byte 2 (if necessary)
public const int GENERIC_SHORT_WRITE = 0x03;

// arg1 = ParamCnt (0, 1, 2)
// arg2 = Parameter byte 1 (if necessary)
// arg3 = Parameter byte 2 (if necessary)
public const int GENERIC_READ = 0x04;

// (see DCS short write commands)
public const int DCS_SHORT_WRITE = 0x5;

// arg1 = MaxReturnPktSize[15:0]
public const int SET_MAX_RETURN_PKT_SIZE = 0x37;

// arg1 = WriteLen
public const int DSI_NULL_PKT = 0x9;

// arg1 = WriteLen
public const int DSI_BLANKING_PKT = 0x19;

// option1: non-zero length payload array
// option2: filename
public const int GENERIC_LONG_WRITE = 0x29;

// (see DCS long write commands)
public const int DCS_LONG_WRITE = 0x39;

// WHEN NOT USED IN MACRO, IMPLEMENTS VIDEO MODE:
// arg1 = FrameCnt
// arg2 = Interlaced (0 == non-interlaced, 1 == interlaced)
//     (arg2 applies to YCBCR formats only)
// option1: non-zero length payload array
//     (length = FrameCnt * HTotal * HVTotal * bytesPerPix)
// option2: filename
//
// WHEN USED IN MACRO, SENDS SINGLE PACKET
// option1: non-zero length payload array
// option2: filename (sends entire file)
// option3: filename = "USERFRAME<n>" where <n> is 0-3
//     If this option is used, internal buffer previously
//     loaded with LOAD_FRAME command is used.
//     arg1 = byte offset in buffer
//     arg2 = payload bytes to use from buffer
public const int PACKED_PIXEL_STREAM_565 = 0x0e;
public const int PACKED_PIXEL_STREAM_666 = 0x1e;
public const int LOOSE_PIXEL_STREAM_666 = 0x2e;
public const int PACKED_PIXEL_STREAM_888 = 0x3e;
public const int LOOSE_PIXEL_STREAM_20_YCBCR_422 = 0x0c;
public const int PACKED_PIXEL_STREAM_24_YCBCR_422 = 0x1c;
public const int PACKED_PIXEL_STREAM_16_YCBCR_422 = 0x2c;
public const int PACKED_PIXEL_STREAM_101010 = 0x0d;
public const int PACKED_PIXEL_STREAM_121212 = 0x1d;
public const int PACKED_PIXEL_STREAM_12_YCBCR_420 = 0x3d;

////////////////////////////////////////////////////
```

```
// CSI commands
//
// Note: the command values defined here are applicable
// when using RPC script files and match the command codes
// in the CSI specification.  On the other hand, when
// using these command codes in RPC calls via the
// PGRemoteRPCClient DLL, all DSI commands will have 0x400
// and all CSI commands will have 0x500 added to them
// (to make them unique across standards).
////////////////////////////////////////////////

// arg1 = FrameNum[15:0]
public const int FRAME_START = 0x0;
public const int FRAME_END = 0x1;

// arg1 = LineNum[15:0]
public const int LINE_START = 0x2;
public const int LINE_END = 0x3;

// arg1 = Value[15:0]
public const int GENERIC_SHORT_PKT1 = 0x8;
public const int GENERIC_SHORT_PKT2 = 0x9;
public const int GENERIC_SHORT_PKT3 = 0xa;
public const int GENERIC_SHORT_PKT4 = 0xb;
public const int GENERIC_SHORT_PKT5 = 0xc;
public const int GENERIC_SHORT_PKT6 = 0xd;
public const int GENERIC_SHORT_PKT7 = 0xe;
public const int GENERIC_SHORT_PKT8 = 0xf;

// arg1 = WriteLen
public const int CSI_NULL_PKT = 0x10;
public const int CSI_BLANKING_PKT = 0x11;

// option1: non-zero length payload array
// option2: filename
public const int LONG_PKT = 0x12;

// WHEN NOT USED IN MACRO, IMPLEMENTS VIDEO MODE:
// arg1 = FrameCnt
// option1: non-zero length payload array
//     (length = FrameCnt * HTotal * HVTotal * bytesPerPix)
// option2: filename
//
// WHEN USED IN MACRO, SENDS SINGLE PACKET
// option1: non-zero length payload array
// option2: filename (sends entire file)
// option3: filename = "USERFRAME<n>" where <n> is 0-3
//     If this option is used, internal buffer previously
//     loaded with LOAD_FRAME command is used.
//     arg1 = byte offset in buffer
//     arg2 = payload bytes to use from buffer
public const int PIXEL_STREAM_YUV420L_8 = 0x1A;
public const int PIXEL_STREAM_YUV422_8 = 0x1E;
public const int PIXEL_STREAM_YUV422_10 = 0x1F;
public const int PIXEL_STREAM_RGB444 = 0x20;
public const int PIXEL_STREAM_RGB555 = 0x21;
public const int PIXEL_STREAM_RGB565 = 0x22;
```

```
public const int PIXEL_STREAM_RGB666 = 0x23;
public const int PIXEL_STREAM_RGB888 = 0x24;
public const int PIXEL_STREAM_RAW6 = 0x28;
public const int PIXEL_STREAM_RAW7 = 0x29;
public const int PIXEL_STREAM_RAW8 = 0x2A;
public const int PIXEL_STREAM_RAW10 = 0x2B;
public const int PIXEL_STREAM_RAW12 = 0x2C;
public const int PIXEL_STREAM_RAW14 = 0x2D;

// option1: non-zero length payload array
// option2: filename
public const int USER_8BIT_TYPE1 = 0x30;
public const int USER_8BIT_TYPE2 = 0x31;
public const int USER_8BIT_TYPE3 = 0x32;
public const int USER_8BIT_TYPE4 = 0x33;

////////////////////////////////////////////////
// DCS_SHORT_WRITE cmd types
////////////////////////////////////////////////

// 0 argument commands
public const int ENTER_IDLE_MODE = 0x39;
public const int ENTER_INVERT_MODE = 0x21;
public const int ENTER_NORMAL_MODE = 0x13;
public const int ENTER_PARTIAL_MODE = 0x12;
public const int ENTER_SLEEP_MODE = 0x10;
public const int EXIT_IDLE_MODE = 0x38;
public const int EXIT_INVERT_MODE = 0x20;
public const int EXIT_SLEEP_MODE = 0x11;
public const int NOP = 0;
public const int SET_DISPLAY_OFF = 0x28;
public const int SET_DISPLAY_ON = 0x29;
public const int SET_TEAR_OFF = 0x34;
public const int SOFT_RESET = 0x01;

// arg1 = Mode[7:0]
public const int SET_ADDRESS_MODE = 0x36;

// arg1 = GammaCurve[7:0]
public const int SET_GAMMA_CURVE = 0x26;

// arg1 = Format[7:0]
public const int SET_PIXEL_FORMAT = 0x3a;

// arg1 = M[0]
public const int SET_TEAR_ON = 0x35;

////////////////////////////////////////////////
// DCS_READ cmd types
////////////////////////////////////////////////

public const int GET_ADDRESS_MODE = 0x0b;
public const int GET_BLUE_CHANNEL = 0x08;
public const int GET_DIAGNOSTIC_RESULT = 0x0f;
public const int GET_DISPLAY_MODE = 0x0d;
public const int GET_GREEN_CHANNEL = 0x07;
public const int GET_PIXEL_FORMAT = 0x0c;
```

```
public const int GET_POWER_MODE = 0x0a;
public const int GET_RED_CHANNEL = 0x06;
public const int GET_SCANLINE = 0x45;
public const int GET_SIGNAL_MODE = 0x0e;
public const int READ_DDB_CONTINUE = 0xa8;
public const int READ_DDB_START = 0xa1;
public const int READ_MEMORY_CONTINUE = 0x3e;
public const int READ_MEMORY_START = 0x2e;

////////////////////////////////////////////
// DCS_LONG_WRITE cmd types
////////////////////////////////////////////

// arg1 = VertScrollPtr[15:0]
public const int SET_SCROLL_START = 0x37;

// arg1 = Line[15:0]
public const int SET_TEAR_SCANLINE = 0x44;

// arg1 = StartColumn[15:0]
// arg2 = EndColumn[15:0]
public const int SET_COLUMN_ADDRESS = 0x2a;

// arg1 = StartPage[15:0]
// arg2 = EndPage[15:0]
public const int SET_PAGE_ADDRESS = 0x2b;

// arg1 = StartRow[15:0]
// arg2 = EndRow[15:0]
public const int SET_PARTIAL_AREA = 0x30;

// arg1 = TopFixedArea[15:0];
// arg2 = VertScrollArea[15:0];
// arg3 = BotFixedArea[15:0];
public const int SET_SCROLL_AREA = 0x33;

// option1: non-zero length payload array
// option2: filename
public const int WRITE_LUT = 0x2d;

// arg1 = FileOffset
// arg2 = WriteLen
// option1: non-zero length payload array
//          (FileOffset and WriteLen are ignored) OR
// option2: non-empty filename
public const int WRITE_MEMORY_CONTINUE = 0x3c;

// arg1 = FileOffset
// arg2 = WriteLen
// option1: non-zero length payload array
//          (FileOffset and WriteLen are ignored) OR
// option2: non-empty filename
public const int WRITE_MEMORY_START = 0x2c;

////////////////////////////////////////////////
// DPHY and generic commands
////////////////////////////////////////////////
```

```
// no arguments
public const int BTA = 0x102;
public const int RESET_TRIGGER = 0x104;
public const int TRIGGER1 = 0x105;
public const int TRIGGER2 = 0x106;
public const int TRIGGER3 = 0x107;

// arg1 = CmdByte[7:0]
public const int ESCAPE = 0x108;

// arg1 = dly in ns (int, will be quant. to 2x LPClock period)
// arg2 = enable drivers (1), tri-state drivers (0)
// arg3 = value (0 == LP00, 1 == LP01, 2 == LP10, 3 == LP11)
public const int LP_DELAY = 0x109;

// no arguments
public const int ENTER_ULPS = 0x10a;
public const int EXIT_ULPS = 0x10b;
public const int WAIT_FOR_BTA = 0x10c;
public const int CLOCK_ON = 0x10d;
public const int CLOCK_OFF = 0x10e;

// arg1 = dly in ns (int, will be quant. to 2x LPClock period)
// arg2 = enable drivers (1), tri-state drivers (0)
// arg3 = bit[1..0]: lane0 (0==LP00, 1==LP01, 2==LP10, 3==LP11)
//        bit[3..2]: lane1 (0==LP00, 1==LP01, 2==LP10, 3==LP11)
//        bit[5..4]: lane2 (0==LP00, 1==LP01, 2==LP10, 3==LP11)
//        bit[7..6]: lane3 (0==LP00, 1==LP01, 2==LP10, 3==LP11)
//        bit[10..8]: clk   (0==LP00, 1==LP01, 2==LP10, 3==LP11,
//                           4==Previous state)
public const int LP_DELAY_ALL_LANES = 0x10f;

// Waits for PG event on event0 line before continuing a macro
// Outputs the specified static LP state while waiting
//
// arg1 = enable drivers (1), tri-state drivers (0)
// arg2 = bit[1..0]: lane0 (0==LP00, 1==LP01, 2==LP10, 3==LP11)
//        bit[3..2]: lane1 (0==LP00, 1==LP01, 2==LP10, 3==LP11)
//        bit[5..4]: lane2 (0==LP00, 1==LP01, 2==LP10, 3==LP11)
//        bit[7..6]: lane3 (0==LP00, 1==LP01, 2==LP10, 3==LP11)
//        bit[10..8]: clk   (0==LP00, 1==LP01, 2==LP10, 3==LP11,
//                           4==Previous state)
public const int WAIT_EXT_EVENT = 0x110;

// EXPERIMENTAL:
// Asserts TrigOut on the PG (NOT the P331/P332 probe) for
// approximately the specified duration (behaves identical
// to LP_DELAY on the MIPI bus).  The width of the pulse
// is nominally specified but may vary based on surrounding
// commands in the macro.
//
// Also, TrigOut on the PG will occur well before the position
// in the macro is is played out of the P331, so it is
// best to insert an LPDelay block preceeding the
// ASSERT_PG_TRIG command to align the PG trigger and P331
// output latency difference.  The command is intended
```

```
// for signals such as reset that do not require exact timing.
//
// Note that latency changes with frequency and should be
// empirically determined.  Estimates for latency are:
// 1.8 us (500 MHz), 2.4 us (300 MHz), 6.9 us (100 MHz)
//
// arg1 = nominal trigger pulse width in ns
// arg2 = enable drivers (1), tri-state drivers (0)
// arg3 = bit[1..0]: lane0 (0==LP00, 1==LP01, 2==LP10, 3==LP11)
//        bit[3..2]: lane1 (0==LP00, 1==LP01, 2==LP10, 3==LP11)
//        bit[5..4]: lane2 (0==LP00, 1==LP01, 2==LP10, 3==LP11)
//        bit[7..6]: lane3 (0==LP00, 1==LP01, 2==LP10, 3==LP11)
//        bit[10..8]: clk   (0==LP00, 1==LP01, 2==LP10, 3==LP11,
//                           4==Previous state)
public const int ASSERT_PG_TRIG = 0x111;

// Defines a command insertion point when used in looping macro
// User must ensure bus is at LP11 upon reaching this command.
// If command is inserted, program is delayed until command
// completes.  If command is not inserted, very short LP11
// period is output on all data lanes.
//
// arg1 = dly in ns (int, will be quant. to 2x LPClock period)
public const int CMD_INSERTION_POINT = 0x112;

// Asserts TrigOut on the P338
// arg1 = trig cmd: 0=lowV, 1=highV, 2=toggleV, 3=pulseV
// Note: Toggle command starts from current voltage
//     (i.e. may be low->high or high->low)
// Note: Pulse command starts from current voltage
//     (i.e. may be low->high->low or high->low->high)
// Use SET_PROBE_TRIG_DURATION command to set the pulse width
public const int ASSERT_PROBE_TRIG = 0x113;

// Asserts LP state (and possibly HS clock state) onto
// the bus until the specified position in the current macro.
// The delay parameter is interpreted relative to the "zero
// position" as indicated by the last call (in the macro) of
// MARK_ZERO_POS.  If there is no last call to MARK_ZERO_POS,
// the start of the macro is used.  The delay is quantized to
// a 4x LPClkFreq boundary.
//
// arg1 = delay in bytes per lane (i.e. 8 HS bits)
// arg2 = enable drivers (1), tri-state drivers (0)
// arg3 = bit[1..0]: lane0 (0==LP00, 1==LP01, 2==LP10, 3==LP11)
//        bit[3..2]: lane1 (0==LP00, 1==LP01, 2==LP10, 3==LP11)
//        bit[5..4]: lane2 (0==LP00, 1==LP01, 2==LP10, 3==LP11)
//        bit[7..6]: lane3 (0==LP00, 1==LP01, 2==LP10, 3==LP11)
//        bit[10..8]: clk   (0==LP00, 1==LP01, 2==LP10, 3==LP11,
//                           4==Previous state)
public const int LP_DELAY_TO_POS = 0x114;

// Records the current byte position in the compile stream
// when building a macro to use as a time reference
// (Subsequent calls to LP_DELAY_TO_POS use this "zero
// position" as the reference to allow exact timing for
// constructing video lines).
```

```
        //
        // no arguments
        // other MIPI parameters (VC, DT_MODE, etc) are ignored
        public const int MARK_ZERO_POS = 0x115;

        // arg1 = ParamCnt (1, 2)
        // arg2 = Parameter byte 1
        // arg3 = Parameter byte 2 (if necessary)
        public const int GENERIC_SHORT_READ_RESPONSE = 0x1f5;

        // option1: non-zero length payload array
        // option2: filename
        public const int GENERIC_LONG_READ_RESPONSE = 0x1f6;

        // arg1 = ParamCnt (1, 2)
        // arg2 = Parameter byte 1
        // arg3 = Parameter byte 2 (if necessary)
        public const int DCS_SHORT_READ_RESPONSE = 0x1f7;

        // option1: non-zero length payload array
        // option2: filename
        public const int DCS_LONG_READ_RESPONSE = 0x1f8;

        // arg1 = ErrorBits[15:0]
        public const int ACK_AND_ERROR_REPORT = 0x1f9;

        // arg1 = DataID
        // non-zero length payload array
        // NOTE: VC arg is ignored for this command
        public const int CUSTOM_LONG_COMMAND = 0x1fa;


        // option1: non-zero length payload array OR
        // option2: filename
        public const int FILE_COMMAND = 0x1fb;

        // arg1 = DataID
        // non-zero length payload array
        // NOTE: VC arg is ignored for this command
        public const int CUSTOM_COMMAND = 0x1fc;

        // arg1: filename
        public const int RPC_SCRIPT = 0x1fd;
    }
}
```